Implementing a Bot Assistant: A case study for a bot helping users using an app

Kalleas Christoforos

SID: 3306170003

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2019

THESSALONIKI – GREECE

Implementing a Bot Assistant: A case study for a bot helping users using an app

Kalleas Christoforos

SID: 3306170003

Supervisor: Dr. Ioannis Magnisalis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2019

THESSALONIKI – GREECE

Abstract

This thesis was written as a part of the MSc in Mobile and Web Computing at the International Hellenic University. Its main scope was to show how a chatbot could act as a virtual assistant, for the users of a large-scale application, providing the following features:

- ✓ Interactive user guidance
- ✓ Modeling and performing business processes
- ✓ Personalized services to the users (i.e. provision of notifications, hints and personal information) based on their past preferences and action history

So, initially, a web application was created to computerize the business processes that take place in a hypothetical medical center. Then, for the sake of this application, a chatbot was created to provide all the above services to the users.

Kalleas Christoforos

Saturday, November 30, 2019

Acknowledgements

This thesis could not be realized without the guidance and the unreserved support of the supervising professor Dr. Ioannis Magnissalis.

Finally, it should be noted the very useful contribution (with comments, suggestions and ideas) of colleagues who also did their thesis under the supervision of Dr. Magnissalis.

Kalleas Christoforos

Saturday, November 30, 2019

Contents

Pictures

Tables

1 Introduction

In the field of software engineering, it is well known that the degree of the user acceptance of an application depends largely on how easily and efficiently users interact with it [1]. Thus, the User Interface (UI) [2], as the part of the application where this interaction takes place, plays a key role in the success of the application.

The most common UI design technique is the Graphical User Interface (GUI) [3]. A GUI consists of visual elements (e.g. text boxes, combo boxes, buttons, images etc.) that allow an application to receive data and, simultaneously, extract information. In fact, the GUI creates the "user experience" within an application [4].

However, as the business complexity and functionality of applications increase, it is quite difficult for the GUI itself to guide and support users as they work on the system [5]. Although a human assistant would be a perfect solution - for this problem - a virtual assistant could be equally helpful. Nowadays, chatbots [6] have been increasingly demanding the role of the virtual assistant in various applications. Based on Artificial Intelligence (AI), they can simplify the interaction between humans and computers using more human-friendly techniques.

1.1 Problem Statement

Typically, a large-scale application has many features in order to meet many functional and non-functional requirements. Therefore, the UI of such an application contains a variety of workplaces that offer various tools and functions within them.

So, as users try to work within the application, they face some needs that cannot be effectively addressed by the UI itself and the usual auxiliary material that comes with the application (e.g. the user manual).

More specifically, these needs are the following:

Need 1. Direct and interactive user guidance

As users try to achieve their business goals, many "how-to" questions usually arise. Traditionally, apart from human help, the only help users can expect - to deal with these questions - comes from the user manual [7].

Usually, the user manual is in the form of a large book full of instructions and process descriptions. However, this cannot be considered as a handy solution and involves only users who are familiar with textbooks and reading instructions. On the contrary, a more interactive method of providing assistance would certainly be more acceptable.

Need 2. Executing business processes in a structured and automated way

In order to execute a business process within an application, users must perform a specific sequence of actions. Typically, in order to do this, they must make the right choices from huge menus and work in complex user interfaces.

Unfortunately, this way of working is very likely to make users feel "lost" and "confused". Instead, they would prefer to follow specific structured workflows to tackle business needs [8]. In addition, they would feel even better if they were guided by the system itself to do so.

Need 3. Personalized services

It is a fact that, when a user has been using an application for a long time, it indirectly builds a kind of personal profile. Usually, it gives the same answers to the same questions and makes the same choices when asked to choose from the same lists of options. In addition, it is likely to provide the application with a lot of personal data.

Therefore, the user would expect from the system to take advantage of its past preferences and action history [9] providing notifications, hints etc. This, of course, would facilitate its work and would provide it with a better user experience.

1.2 Objectives and Aims

It is obvious that a skilled person, who can always be next to the user, could help user to cope with many of the above needs. However, of course, this is practically impossible.

So, the aim of this thesis is to show that a virtual assistant, having the form of a chatbot, can be a satisfactory solution to the problem discussed above.

More specifically, the detailed objectives of this thesis are as follows:

Objective 1. Direct and interactive user guidance

To show how a chatbot, acting as a virtual assistant, can provide interactive guidance to the users of an application and, consequently, answer their "how-to" questions.

Objective 2. Executing business processes in a structured and automated way

To show how a chatbot, acting as a virtual assistant, can provide straight and structured workflows for the users to execute specific tasks.

Objective 3. Personalized services

To show how a chatbot, acting as a virtual assistant, can provide - whenever possible – personalized services (i.e. provision of notifications, hints and personal information) to the users, based on past user preferences and action history.

2 Literature Review

This chapter defines the theoretical and academic background of this thesis. It refers to other related work and identifies the particularities and challenges that affect the achievement of the thesis objectives.

2.1 User Guidance

Even the most advanced users of a large application need help and guidance to perform various tasks. Traditionally, in software engineering industry, the most common way of providing help to the users - of an application - is the *user manual* [7].

The user manual is intended to guide users in performing tasks. Therefore, it covers all the processes that can be performed within the application by providing appropriate instructions. In addition, it may include a "how-to" section to provide information on how various business needs can be addressed by the application.

Usually, user manuals look like books, either in print or electronic form (e.g. PDFs, HTML files etc.), containing static text and images. Thus, for every need, the user has to search - within the manual – for the relevant section and, then, study its contents.

2.2 Business Process Modeling and Performing

In the field of software engineering, *business process modeling* [10] is a modern process automation methodology for applications. Its main objective is to model the business processes that can be executed inside the application and define specific workflows for the execution of them.

Subsequently, the users of the application can achieve any business goal by following the corresponding workflow. Thus, they can be significantly more productive [8], especially when it comes to achieving important and critical business objectives.

2.2.1 Process Modeling Techniques

There are indeed many techniques for modeling the business processes that can be executed within an application. One of the most common techniques is Business Process Model and Notation (BPMN) [11]. It provides a graphical notation for modeling business processes even if they are complex and involving more than one user.

After modeling, special BPM engines (i.e. special software) can execute the business models. So, if a user wants to execute a specific business process it has only to execute the corresponding model through a BPM engine. During the model execution, the system itself guides the user to execute specific actions.

2.3 Personalized Services in Applications

Personalized user services are becoming increasingly popular in software engineering as - it seems that - they promote user satisfaction [9]. According to related theories, user satisfaction increases when the content suggested to them – by an application - matches their interests. At the same time, it is evident that users prefer content recommended by a process in which they are directly involved.

However, research examining the provision of personalized services is still relatively inadequate [12]. Obviously, when an application requires its users to explicitly specify their preferences, it is relatively easy to provide such services. The challenge, of course, is to have them automatically generated, based only on user action history and past user preferences.

2.4 Virtual Assistants for Applications

A *virtual assistant* is a specific kind of software that has the role to assist the users of an application to use the variety of tools and functions that the application provides [13]. In most cases, a virtual assistant uses Artificial Intelligence (AI) techniques and can communicate with users by understanding free text and speech.

2.4.1 Assistant Services

Based on the experience of recent years, the assistant services that virtual assistants can offer can be classified into five levels [14] depending on how much intelligence is required to provide them. The services of the first level require the least intelligence while the services of the fifth level require the greatest one.

These levels are the following:

Level 1. Notification Services

They provide users with information about events that have taken place inside the application domain and are related to them. Usually, this information has the form of text messages.

Level 2. How-to Services

They provide users with help in performing specific tasks. But unlike the usual FAQ (Frequently Asked Questions) tools, they are intended to describe the required actions in detail.

Level 3. Contextual Services

These are services that are based on recognizing and understanding the context of a conversation. So, the meaning of what the user says is of particular importance. At the same time, given a specific thematic context, the virtual assistant can understand irrelevant inputs and react properly.

Level 4. Personalized Services

They are based on past user preferences and history of user actions. They provide users with personalized suggestions, notifications and personal information. In a sense, therefore, they provide a fully personalized work experience.

Level 5. Autonomous Services

These are services that are, first of all, fully customizable to the users. In providing them, the virtual assistant performs various tasks, on behalf of the users, requiring them to have little or no involvement.

2.5 Chatbots

A *chatbot* (or, simply, *bot*) is a computer program, based on Artificial Intelligence (AI), which interacts with users in natural language [6]. In general, a chatbot can be considered as a question-response system designed to simulate a smart conversation with a human partner. This ability is based on all the recent developments in the fields of Natural Language Understanding (NLU) [15] and Voice Recognition Technology (VRT) [16].

2.5.1 History of Chatbots

The first chatbots were not really smart, but they actually had a collection of predefined answers that matched specific questions. They were elementary and were trying to create the spoofing of a conversation between human and computer. In addition, they had little to no contextual understanding [17].

The principles of chatbots are based on Alan Turing's "Computing Machinery and Intelligence" paper in 1950. In particular, the "Turing test", developed on this paper, is widely regarded as the key criterion for assessing the intelligence of an electronic system [18].

One of the most important first attempts to implement the "Turing test" is considered the ELIZA program developed in 1966 at MIT [19]. ELIZA provided the user with the possibility of a simple conversation and worked very actively in the scientific community.

2.5.2 Chatbots and Interactional Capability

In order to meet the user expectations for human-like interaction, chatbots must address the concept of *interactional capability*. Interactional capability exceeds technical capabilities and has the meaning of reaching a rather communication goal [20].

Interactive capability is formed by a set of features [21] that allows a chatbot to participate actively in a conversation and show awareness of the subject being discussed, the evolving chat context and the flow of dialogue.

Contagiousness

At first, *contagiousness* is defined as the feature of a chatbot to communicate to users the underlying logic [22]. Providing contagiousness helps users to identify the possibilities embedded in the software [23], which improves system usability [24].

Contagiousness is, in other words, the capability of a chatbot to communicate the system features to users [25]. The interesting part of this communication lies in the nature of the interface used. Thus, instead of menus and buttons, chatbots reveal the system's capabilities through conversation, bringing new ways to the learning of the system.

Conceptuality

Conceptuality is a chatbot feature that enables it to prove its attention in a conversation [26]. In particular, it allows it to monitor the flow of the conversation, to understand its operational context and to interpret every element that arises during its course [27].

Adaptability

In general, *adaptability* refers to the ability to adapt functionality and communication ways, with an individual (or a category of people), to the particular circumstances and characteristics of the individual [28]. With regard to chatbots, adaptability can increase their "social intelligence" [21] [29] by allowing - a chatbot - to realize the specifics of the current situation. At the same time, it provides the ability – for the chatbot - to adapt dynamically its "behavior" to better respond to special needs [30].

Proactivity

In general, *proactivity* [21] refers to the ability to act autonomously on behalf of the user [31], thereby reducing the effort required to perform a task [32]. Regarding chatbots, proactivity allows a chatbot to take initiatives during the execution of a task [27]. In particular, chatbots, based on proactivity, can propose problem-solving methods or provide new data.

2.5.3 Chatbot Technologies

The above capabilities have emerged as a result of the application of various technologies.

Automatic Speech Recognition (ASR)

Speech recognition is one of the most revolutionary techniques in human-computer interaction [33]. It has only been possible in recent decades as it requires high computing power and storage capacity.

Automatic Speech Recognition (ASR) is, actually, one of the speech recognition phases. It aims to convert speech to text while preserving speech context [33] [34]. After that, other language processing technologies can be used to analyze and process the resulting text.

Natural Language Processing (NLP)

Natural Language Processing (NLP) deals with a wide variety of issues related to the analysis and computation of human languages [35]. It is an area of Artificial Intelligence and Linguistics that aims to enable computers to understand human languages [36].

Chatbots are, actually, an application area of the NLP technology [33]. They use NLP-based methods to convert natural language text into a programming-friendly data structure, while preserving the meaning of the original text. At the same time, they are able to identify important pieces of information - inside text - such as names, places, events, dates, times and prices [36].

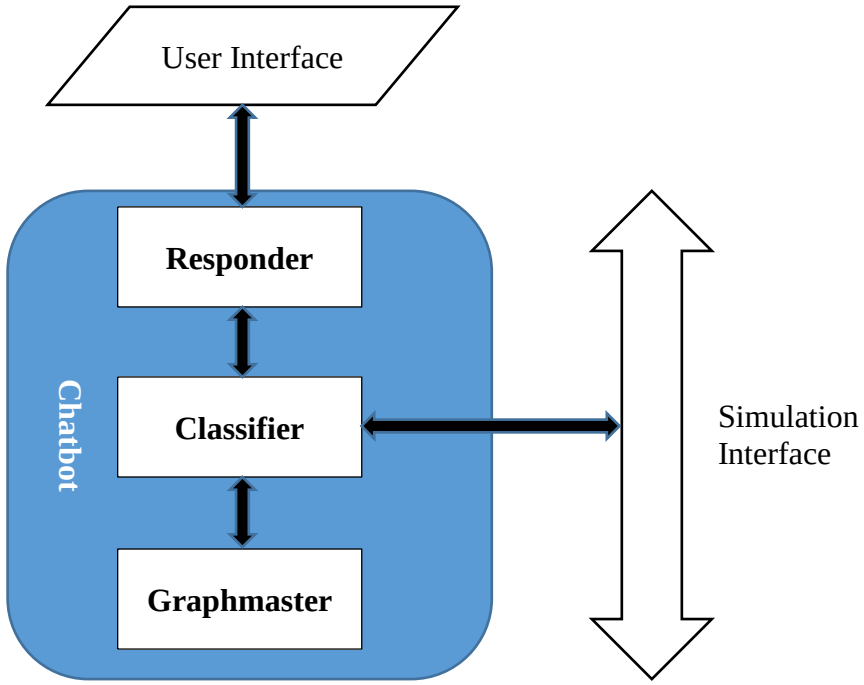
Natural Language Toolkit (NLTK)

The Natural Language Toolkit (NLTK) is a set of tools based on the principles of NLP technology [33]. Specifically, it is used to extract words from text, assemble them into phrases and analyze them semantically.

In fact, NLTK is a set of open source modules. They are written in the Python programming language and are part of the Python libraries.

2.5.4 Chatbot Architecture

Each chatbot is, in fact, a complex software made up of specific components that work together based on a specific architecture [37] [33] (see Picture 1). It is the responsibility of the chatbot developer to implement them and they are as follows:



Picture 1: Chatbot Architecture

- **Responder**

  It is the component that transfers the data, which the user enters, into the internal parts of the chatbot. At the same time, it performs the reverse operation by exporting the chatbot response to the user interface.

- **Classifier**

  At first, it filters and, possibly, corrects the input data. Then it extracts the interesting building elements and, finally, transfers them to the Graphmaster for processing.

  Conversely, it receives the results from the Graphmaster and, after processing, it forwards them to the Responder.

- **Graphmaster**

  It is, in fact, the core component of the chatbot. Applying specific algorithms and based on various models it performs the conceptual analysis of the content.

2.5.5 Human-Chatbot Interaction Process

In general, if the medium of the communication is text, the human-chatbot interaction takes place as a repetitive process of the following two phases [33] [38] (see Picture 2):

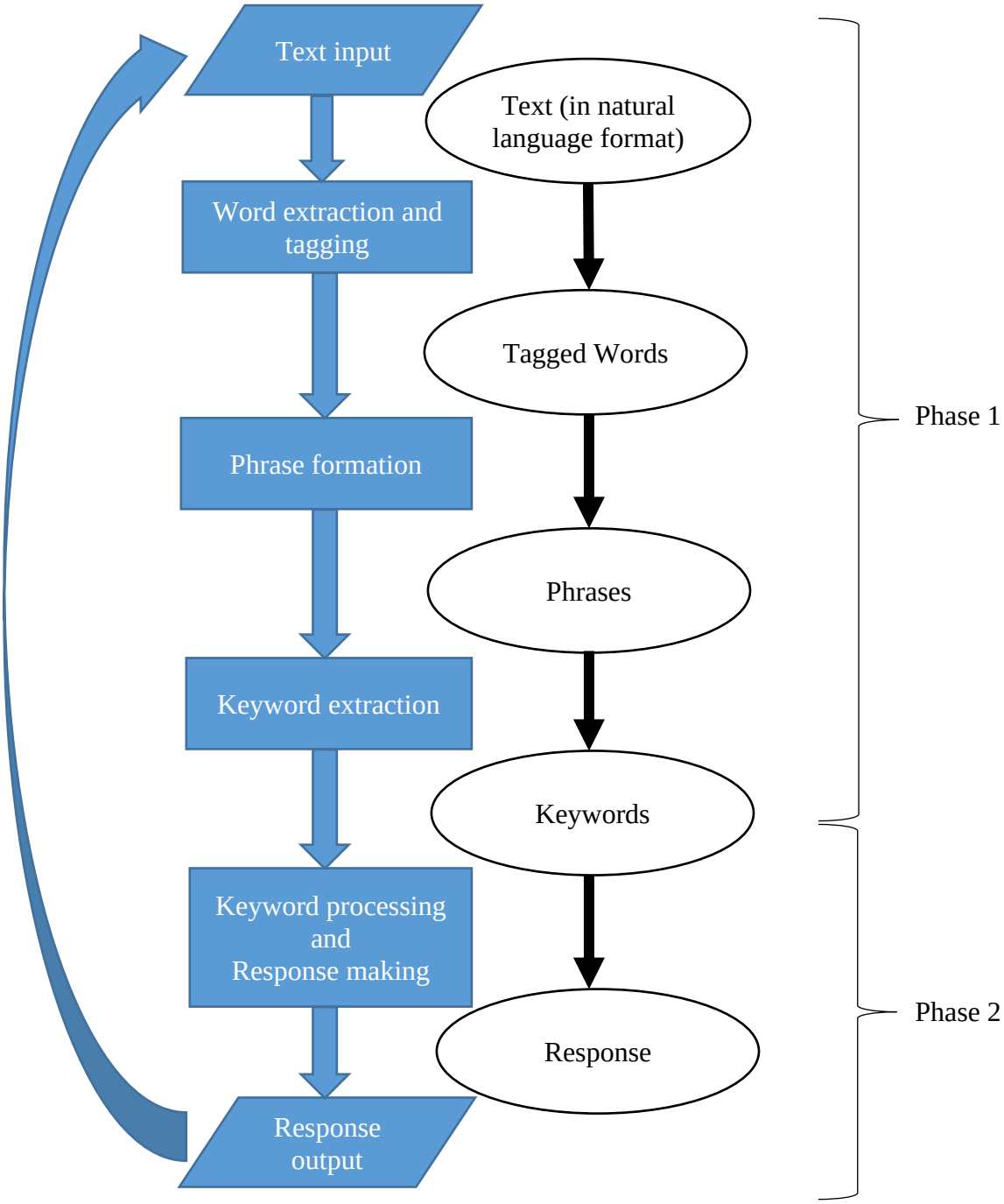
Phase 1. Text processing

Text (in natural language format) is processed, by the chatbot, and the relevant keywords are extracted.

Phase 2. Keywords processing and Response

The extracted – from the previous phase – keywords are processed, by the chatbot, and the adequate response is generated.

Obviously, if speech becomes the medium of human-chatbot communication then, at the beginning of the above process, a voice-to-text conversion phase must be added.



Picture 2: Interaction Process between Human and Chatbot

Text processing

During this phase [33], the inserted text (in natural language format) is processed by the chatbot and the relevant keywords are extracted. The phase consists of the following steps:

Step 1. Text input

Text is inserted by human in natural language format.

Step 2. Word extraction and tagging

The inserted text is parsed into separate words and each word is tagged by its position and its relation to the other words.

Step 3. Phrase formation

Using grammar rules, tagged words form phrases.

Step 4. Keyword extraction

Keywords are extracted from the above phrases by removing insignificant words.

Keyword processing and Response

During this phase [33], the keywords, which have been extracted from the previous phase, are processed by the chatbot and the adequate response is generated. The phase consists of the following steps:

Step 1. Keyword processing and Response making

The chatbot, based on the extracted keywords and its programmable logic, generates the adequate response. The response could be text, speech or the execution of an action.

Step 2. Response output

The generated response is presented to human.

2.5.6 Chatbot User Interface

According to human-chatbot interaction process, any chatbot needs a User Interface that:

- ✓ provides UI elements to human to create input (i.e. a human's working area)
- ✓ presents output to human

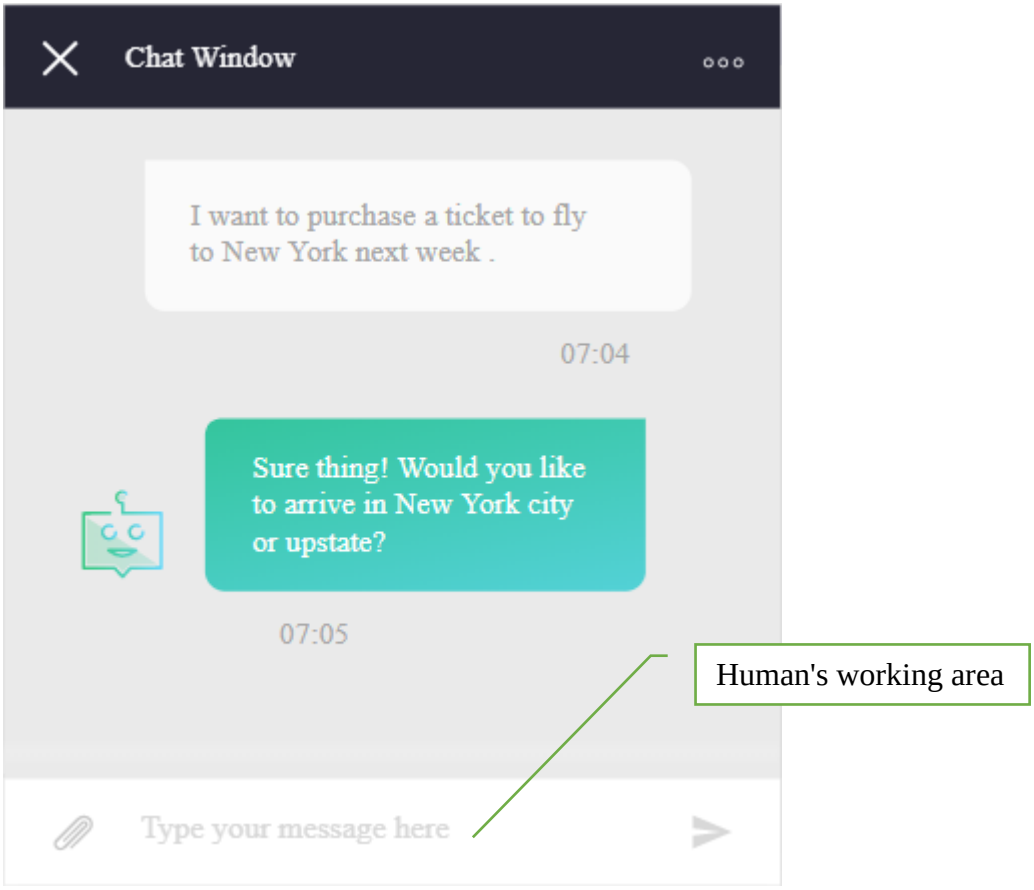
In general, nowadays, there are two distinct trends in designing user interfaces for chatbots [39]:

- **Monolithic design style**
- **Interactive design style**

Monolithic design style

Chatbots that follow the *monolithic design style* are built to provide short answers to short questions. So, they are usually used as customer-service tools for business.

In most cases, they have a simple user interface in which the human's working area consists of a simple text box located at a specific fixed position. A typical example of this category is "Bold 360" [40] (see Picture 3) that can provide a variety of customer services to business.

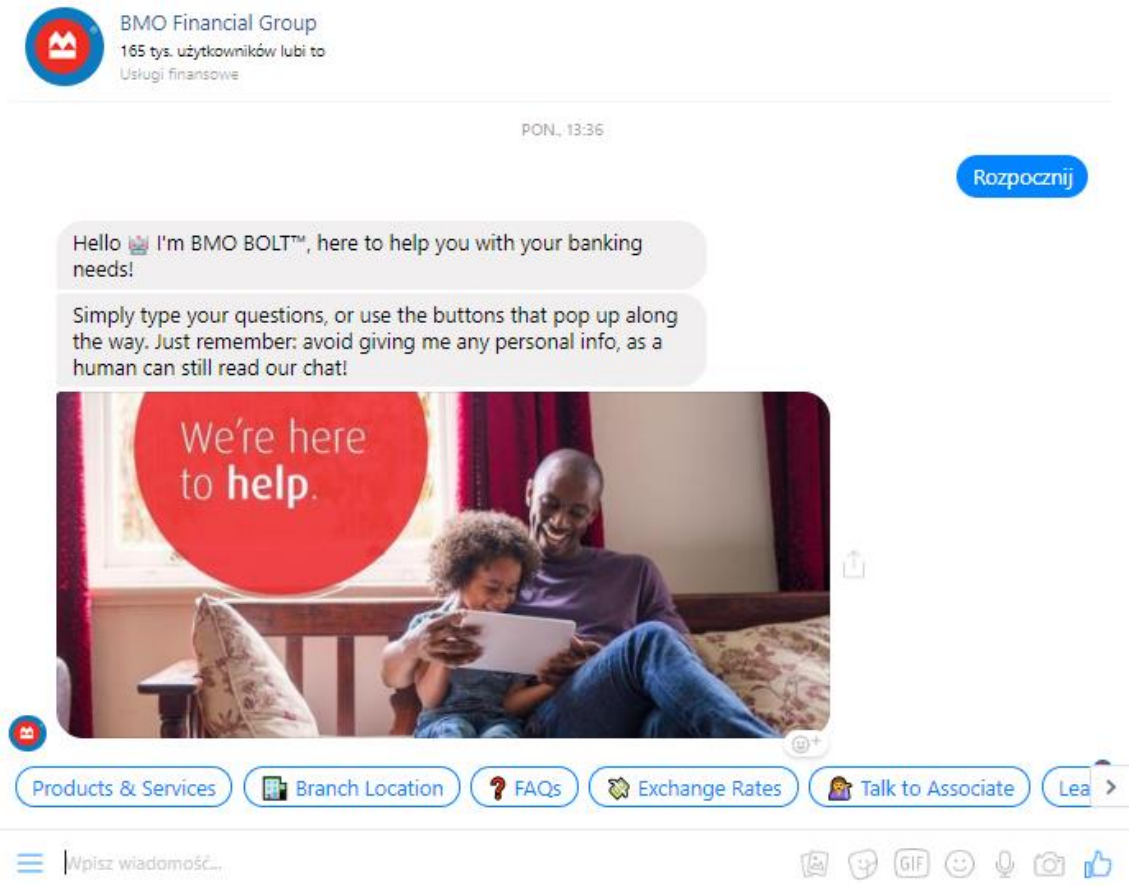


Picture 3: "Bold 360" chatbot

Interactive design style

Chatbots that follow the *interactive design style* try to create a more human-like experience to the human-chatbot conversation. They are usually used to provide an alternative way of interacting with a business for purposes beyond answering simple questions.

In most cases, the human's working area is dynamic in terms of both location and content. Therefore, users can interact with the chatbot either by typing text or using visual UI elements (e.g. Text Boxes, Buttons and Dropdown Lists). A typical example of this category is "BMO Bolt" [41] (see Picture 4) that offers bank services to the customers of Bank of Montreal.



Picture 4: "BMO Bolt" chatbot

2.5.7 Implementing Chatbots

The core engine of any chatbot is the Natural Language Processing Engine (NLP Engine) [17]. Thus, whenever a user makes a query, the chatbot sends it to the NLP Engine. The NLP extracts all the useful query elements and returns them to the chatbot. Finally, the chatbot processes the elements, formulates a respond and returns it to the user.

Implementation techniques

A widely accepted chatbot implementation technique is the *domain-based chatbot* [17]. Chatbots of this category focus on a specific domain and can be used in various areas such as education, customer service, e-commerce and more. According to this approach, user queries are semantically analyzed and transferred to the core component of the system called *domain browser*. The browser finds the appropriate answer to a domain warehouse and eventually returns to a natural language to the user. It has also been found that the efficiency of such a chatbot is increased if the domain knowledge combined with interactive knowledge [42].

Another approach of implementing a chatbot is the *OCR based chatbot* [17]. Chatbots of this approach use the Optical Character Recognition (OCR) technology [43] to convert scanned documents into machine encoded text. Then, from the extracted text, they generate question-answer pairs via transformations and ranking algorithms [44]. Finally, they store the question-answer pairs, as the chatbot knowledge, using AIML [45]. These chatbots can be used in customer services and in education for answering frequently asked questions [46].

Recent known attempts

In general, chatbots can be categorized into two types: *owned* and *shared* [47].

Owned chatbots are being developed by large companies to improve the quality of customer services and reduce the overall cost of these services. This is often the case in sectors like banking, telecommunications and e-commerce. An example is Erica [48] which is constructed by the Bank of America to help customers with bank problems. Another example is Charlie [49], the AT&T support chatbot that acts as customer service representative.

Shared chatbots are, actually, chatbot frameworks that help developers to build their one chatbots. Some examples are Microsoft Bot Framework [50], Facebook Messenger [51], Google Assistant [52], and Amazon Lex [53]. All of these frameworks can be used to build chatbots for various purposes that handle a variety of third-party data.

2.6 Making Chatbots Work as Virtual Assistants

Based on its interactional capability, a chatbot can provide some very useful offerings to the users of an application. In fact, chatbots seems like a perfect choice as virtual assistants in various industry sectors including customer services, health, travel and education [54]. For this reason, more and more businesses are predicting the use of chatbot to serve their customers [47].

2.6.1 How Chatbots can Provide Assistant Services

In general, to be considered a chatbot as a virtual assistant is sufficient to satisfy the levels of assistant services categorization that presented in Section 2.4.1. Chatbots that have reached the highest level can be considered the most sophisticated.

The chatbot that is going to be built for the needs of this thesis is enough to cover the first four levels of assistant services categorization.

Notification assistant services

Using their UI environment, chatbots can easily display various messages to the user. A notification is nothing more than a category of messages.

Of course, the most important element of a notification is its content. The more personal this is, the more important it can be considered [55]. Personalizing notifications is an issue that can be integrated into the broader chapter of providing personalized services (see the discussion about personalized services below).

How-to assistant services

With their interaction capability, chatbots could provide ongoing assistance to the users of an application [25]. Providing customer support [56] services can guide the users to use the UI of an application while executing tasks. The important thing is that they can do this in an interactive way rather than simply by offering a sterile help text [57]. Additionally, chatbots can provide a better user experience making the interaction more human-like [58].

Contextual assistant services

When performing a task, a chatbot could understand the meaning of the interaction and try to guide the entire process efficiently and productively [59]. It is for sure that one of the main reasons to use a chatbot is to increase productivity [60]. Using a chatbot can be more productive than using the tools of a classic user interface (e.g. menus, screens, toolbars etc.) [20]. In any case, it is very helpful for the user to know the reaming steps of a process or the conceptual meaning of each step [61].

Personalized assistant services

The exploitation of personal data from chatbots could certainly foster a relationship of trust and cooperation with the user [62]. Thus, chatbots could customize responses based on user level, personal interests and needs [63]. In any case, it would be ideal for a chatbot to show different behaviors to different users [55].

In addition, in some cases, chatbots could act proactively in order to improve productivity [20]. In this respect, chatbots could make initiatives by asking – the user - appropriate questions [20]. This would reduce the amount of time required for the job and the required effort.

Finally, chatbots could guide users to set goals and track their progress [64]. In addition, they could drive the user to exploit some capabilities of the application, which would otherwise not be willing to do so [65].

2.6.2 Challenges

In recent years, chatbots have evolved to a considerable extent. However, they still do not fully meet user expectations [66] [67] because, while their functional performance has improved considerably [68] [69], their communication skills [20] [70] have not developed to the same extent. This drawback becomes more important since, according to the theory of Equation of the Media [71], people are very interested in the communication characteristics of a computing system [72].

Since chatbots communicate with users interactively, the social factor plays a very important role in their success [66]. Consequently, the issue of user acceptance is mainly social rather than technical [30]. In fact, chatbots demonstrating communication skills are more acceptable than others [73] [74] [75]. On the contrary, if they do not meet these requirements, they may cause frustration and / or discomfort [67].

In addition, most chatbots are built on older human conversations that experience adaptability and privacy problems. Most of the time, users end up waiting for human help to resolve their issue after chatting with a chatbot [47].

3 Problem Definition/Materials & Methods

This chapter sets out the context of this thesis, according to the problem statement (see Section 1.1) and the objectives (see Section 1.2) presented in the introduction. It also defines the basic principles and functionality of the virtual assistant to be built.

3.1 General Scope

The scope of this thesis is generally defined by the "problem statement" described in Section 1.1. Actually, it will try to propose a solution to this problem demonstrating that, given any large-scale application, a chatbot can be created to act as a "virtual assistant" for the users of the application and, thus, meet their needs. At the same time, it intends to show that a chatbot can also be a flexible and satisfying add-on to the application.

Thus, as part of this thesis, a chatbot will be created to provide assistant services to the users of a specific real application. Actually, it is a web application that will be made just for this purpose. It is called "MEdical Center Information System" (or, simply, MECIS) and it is supposed to meet the basic IT needs of a hypothetical medical center.

Finally, the desired chatbot will be developed as a distinct subsystem of the MECIS application. Just for reference purposes, the chatbot will have the code name "MECIS-Bot".

3.2 General Goals

In general, the chatbot to be constructed for this thesis will attempt to cover the first four levels of the assistant services classification presented in Section 2.4.1. At the same time, it will try to meet all the challenges that chatbots usually face when acting as virtual assistants (see Section 2.6.2).

More specifically, this thesis will try to succeed the following goals.

3.2.1 Providing Interactive User Guidance with Chatbot

To build a chatbot that offers a "live" and interactive manual to the users of the MECIS application.

Usually, most of the help an application can provide to its users comes from the user manual. In this thesis, however, an attempt will be made to show how a chatbot can be used, in the alternative, for the same purpose.

3.2.2 Modeling and Performing Business Processes with Chatbot

To build a chatbot that provides straight and structured workflows for the users of the MECIS application, to execute specific business processes.

In fact, based on the literature, chatbots are not suggested as a method for modeling business processes. In this thesis, however, an attempt will be made to use the chatbot's potential to achieve this goal.

3.2.3 Providing Personalized Services with Chatbot

To build a chatbot that provides personalized services (i.e. provision of notifications, hints and personal information), to the users of the MECIS application.

The critical point here is that the chatbot will be based only on stored personal data, past user preferences and the action history.

3.3 The MECIS Application – A Complete Overview

The "MEdical Center Information System" (MECIS) is a complete application that intends to fulfil the basic computing needs of a hypothetical medical center. It is aimed at both patients and staff of the medical center.

3.3.1 Scope

In general, a medical center provides health services to the public. It usually has many doctors with many specialties. In addition, it has auxiliary staff for secretarial support.

Patients who want to visit a doctor must make an appointment first. Needless to say that the medical center has to provide all the necessary information about its doctors and their working schedule.

3.3.2 Stakeholders

The main stakeholders of MECIS are:

M-S1. Employees

All the secretarial support staff and the system administrators.

M-S2. Patients

All the patients of the medical center.

For the sake of simplicity, doctors are not regarded as main stakeholders but as secondary.

3.3.3 Problems

MECIS focuses only on the major problems of the above stakeholders. At the same time, it makes many abstractions to them, as it is only a "case study" and not a commercial information system.

Employees

M-P1. Doctors data management

M-P2. Doctors schedule management

M-P3. Appointment management

Patients

M-P4. Contact the medical center

M-P5. Looking for doctors (of a specific specialty or not)

M-P6. Appointment making

M-P7. Appointment cancellation

M-P8. Overview of appointment history

3.3.4 System Users

MECIS defines three major user roles, which are:

M-U1. Administrator

M-U2. Secretary

M-U3. Patient

Description and responsibilities

Each user, based on its role, gains specific rights and responsibilities (see Table 1).

User Role	Description	Responsibilities
Administrator	Represents the system administrators	- Manages system users
Secretary	Represents all the secretarial support staff	- Feeds the system with the doctor schedule and other useful data (e.g. contact info) - Checks for new appointments and accepts them
Patient	Represents all the patients	- Looks for doctors - Makes appointments

Table 1: Description and responsibilities of MECIS user roles

User working environment

The user working environment is common to all users. Of course, the system provides a specific set of features for each user role.

In addition, in order for a user to work on the system, it must first sign in.

3.3.5 System Functional Requirements

Below, are the functional requirements that MECIS is required to meet.

General requirements

M-FR1. Sign In

Signing into the system, as an active system User.

M-FR2. Sign Up

Singing up to the system as creating a new User of the "PATIENT" Role.

a) The system does not allow, for the new User, to have the same "Username" with someone else user.

- *Any User of the other two roles can be created only by an administrator.*

M-FR3. Sign out

Signing out of the system, after having signed in first.

M-FR4. Open User Profile

Opening the Profile of the current signed in User.

a) The system does not display the User's Password.

M-FR5. Modify User Profile

Modifying the Profile of the current signed in User, after opening it first.

a) The system does not allow, for the current User, to change the "Username".

M-FR6. Change Password

Changing the Password of the current signed in User.

Administrator Requirements

E-Mail subsystem

M-FR7. Set "E-mail Subsystem" properties (e.g. E-mail Server Name, E-mail Address)

- *"E-mail Subsystem" is a subsystem that is responsible for sending e-mail messages to users.*

System users

M-FR8. Create System User

Inserting a new System User to the database.

a) The system does not allow, for the new User, to have the same Username with someone else user.

b) The system sets the Password of the new user equal to its Username.

M-FR9. Browse System Users

Displaying the list of all System Users.

M-FR10. Search for System Users (using criteria upon one or more fields)

Filtering the above list in order to display only the System Users that satisfy specific search criteria.

M-FR11. Open System User

Opening a System User, to display him (or her) data, after selecting him (or her) from the System Users list.

a) The system does not display the User's Password.

M-FR12. Modify System User

Modifying a System User's data, after opening him (or her) first.

a) The system does not allow modifying Username.

M-FR13. Deactivate/Activate System User

Deactivating an active System User (or activating a deactivated System User), after opening him (or her) first.

- *A deactivated System User is not allowed to sign in, anymore.*

Secretary Requirements[1]

Contact info

M-FR14. Open Contact Info

Opening Contact Info data.

M-FR15. Modify Contact Info

Modifying Contact Info data, after opening them first.

Doctors

M-FR16. Create Doctor

Inserting a Doctor to the database.

M-FR17. Browse Doctors

Displaying the list of all Doctors.

M-FR18. Search for Doctors (using criteria upon one or more fields)

Filtering the above list in order to display only the Doctors that satisfy specific search criteria.

M-FR19. Open Doctor

Opening a Doctor, to display him (or her) data, after selecting him (or her) from the Doctors list.

M-FR20. Modify Doctor

Modifying a Doctor's data, after opening him (or her) first.

M-FR21. Write down Doctor Working Hours

Writing down, in a week base, a Doctor's Working Hours, after selecting him (or her) from the Doctors list.

M-FR22. Delete Doctor

Physically removing a Doctor from the database, after selecting him (or her) from the Doctors list.

[1] All Secretary requirements are valid for administrators too.

Appointments

M-FR23. Create Appointment

Inserting an Appointment to the database, on behalf of a specific patient.

> *a) The system does not allow the appointment of non-working hours.*
> *b) The system does not allow the new Appointment to have the same day and time with another Appointment for the same Doctor.*

M-FR24. Browse Appointments

Displaying the list of all Appointments.

M-FR25. Search for Appointments (using criteria upon one or more fields)

Filtering the above list in order to display only the Appointments that satisfy specific search criteria.

M-FR26. Open Appointment

Opening an Appointment, to display its data, after selecting it from the Appointments list.

M-FR27. Modify Appointment

Modifying an Appointment's data, after opening it first.

> *a) The system does not allow the Appointment to have the same day and time with another Appointment for the same Doctor.*

M-FR28. Delete Appointment

Physically removing a pending Appointment from the database, after selecting it from the Appointments list.

> *a) The system allows deleting only pending Appointments that has not been inserted by patients.*

M-FR29. Accept/Reject Appointment

Modifying the state of a pending (and only pending) Appointment to "ACCEPTED" or "REJECTED", after opening it first.

> *a) The E-Mail Subsystem sends an automated e-mail message to the patient's email address, to inform him (or her).*

M-FR30. Cancel Appointment

Canceling a pending or accepted Appointment, after opening it first.

> *a) The E-Mail Subsystem sends an automated e-mail message to the patient's email address, to inform him (or her).*

M-FR31. Close accepted Appointment (as completed)

Closing a previously accepted Appointment as if it has been completed, after opening it first.

> *a) The system changes the Appointment's state to "COMPLETED".*
> *b) The E-Mail Subsystem sends an automated e-mail message to the patient's email address, to inform him (or her).*

Patient Requirements

Contact info

M-FR32. View Contact Info

Viewing Contact Info data.

Doctors

M-FR33. Browse Doctors

Displaying the list of all Doctors.

M-FR34. Search for Doctors (using criteria upon one or more fields)

Filtering the above list in order to display only the Doctors that satisfy specific search criteria.

M-FR35. Search for available Doctors (efficient searching)

Producing a list that contains only the Doctors that have a given specialty and are available during a given period.

> *a) The system displays the list of available doctors sorted based on the selection history. This means that the doctor that has been selected the most – in the past – comes first, and so on.*

M-FR36. View Doctor

Opening a Doctor, to view his (or her) data, after selecting him (or her) from the Doctors list.

Appointments

M-FR37. Create Appointment

Inserting an Appointment to the database, on behalf of the signed in user-patient.

a) The system sets the state of the new Appointment to "PENDING".
b) The system does not allow the appointment of non-working hours.
c) The system does not allow the new Appointment to have the same day and time with another Appointment for the same Doctor.

M-FR38. Browse Appointments

Displaying the list of all Appointments that have been inserted by the signed in user-patient.

M-FR39. Search for Appointments (using criteria upon one or more fields)

Filtering the above list in order to display only the Appointments that satisfy specific search criteria.

M-FR40. Open Appointment

Opening an Appointment, to display its data, after selecting it from the Appointments list.

M-FR41. Modify Appointment

Modifying a pending (and only pending) Appointment's data, after opening it first.

a) The system does not allow, for a patient, to change the Appointment's state.

M-FR42. Cancel Appointment

Canceling a pending or accepted Appointment, after opening it first.

a) The E-Mail Subsystem sends an automated e-mail message to the medical center's email address (as it has been defined in the contact info).

3.3.6 System Non-Functional Requirements

Below, are the non-functional requirements that MECIS is required to meet.

Applicable standards

M-NFR1. 3-tier architecture

MECIS is a Web Application that based on 3-tier architecture. It uses PHP for the backend and HMTL/JAVASCRIPT for the front.

System requirements

M-NFR2. Apache Web Server

MECIS runs on an Apache Web Server.

M-NFR3. MySQL Database

MECIS stores its data on a MySQL Database.

Design constraint requirements

M-DR1. Common browsing mechanism

All Lists are displayed in browsers that have the same structure, same layout and same (basic) functionality.

M-DR2. Common filtering mechanism

All browsers offer the same basic filtering mechanism.

M-DR3. Common Opening mechanism

All entities are opened in *editors* that have the same structure, same layout and same (basic) functionality.

M-DR4. Common deleting mechanism

All browsers offer the ability to delete a record using the same mechanism.

Logical database requirements

M-LDR1. Referential Integrity

MECIS does not allow:

1) Deleting a Doctor that has related Appointments.
2) Inserting an Appointment without defining the related Doctor and user-patient.

3.3.7 System Modeling

In general, MECIS is based on the 3-tier architectural model and the MVC [76] design pattern.

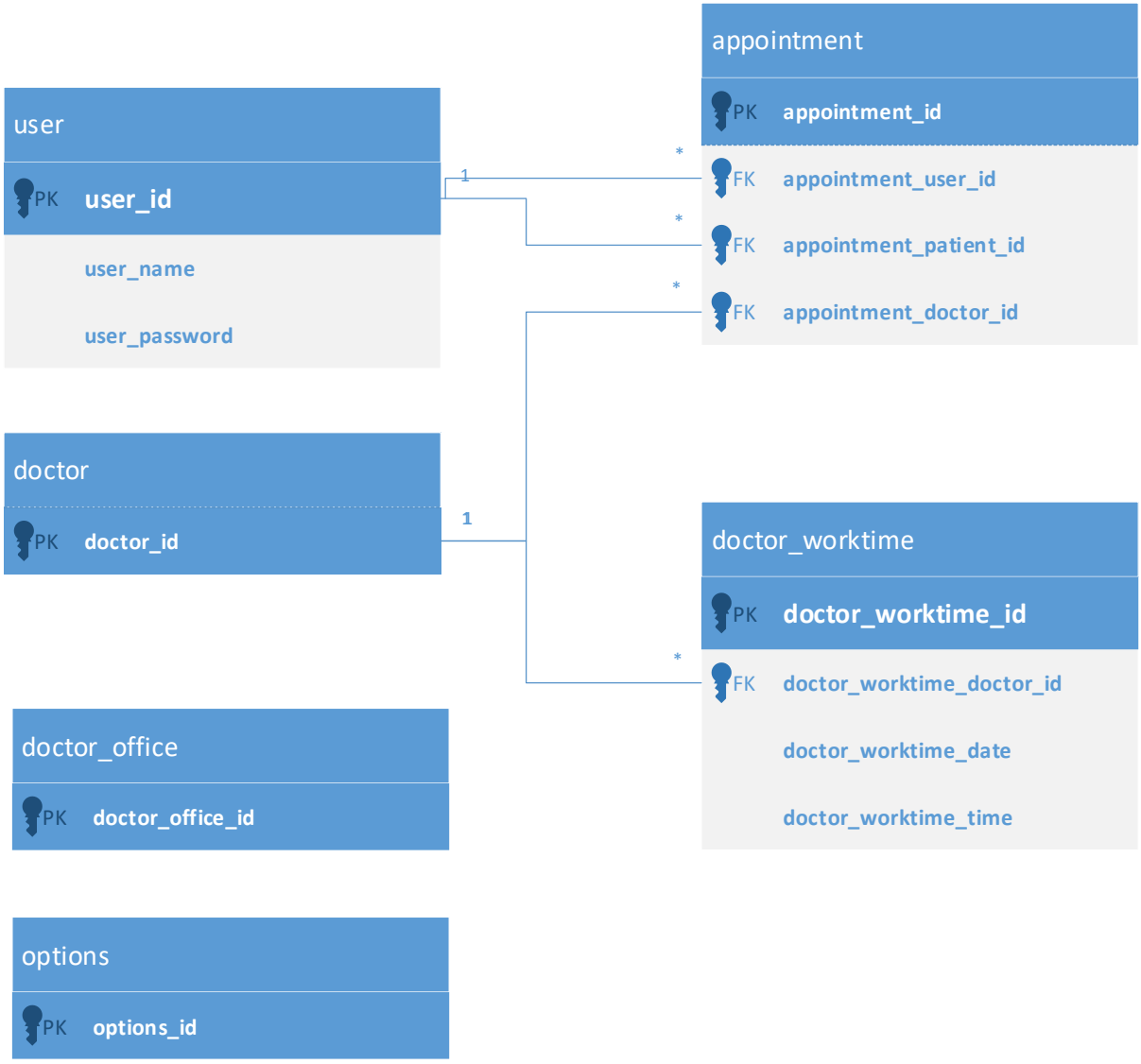
General Architecture

The MECIS system is subdivided into the following three tiers:

- **MECIS Database Server**
- **MECIS Application Server**
- **MECIS Client**

MECIS Database Server

The MECIS Database Server contains the relational database that stores all data of MECIS. The ERD diagram of the database is given below.

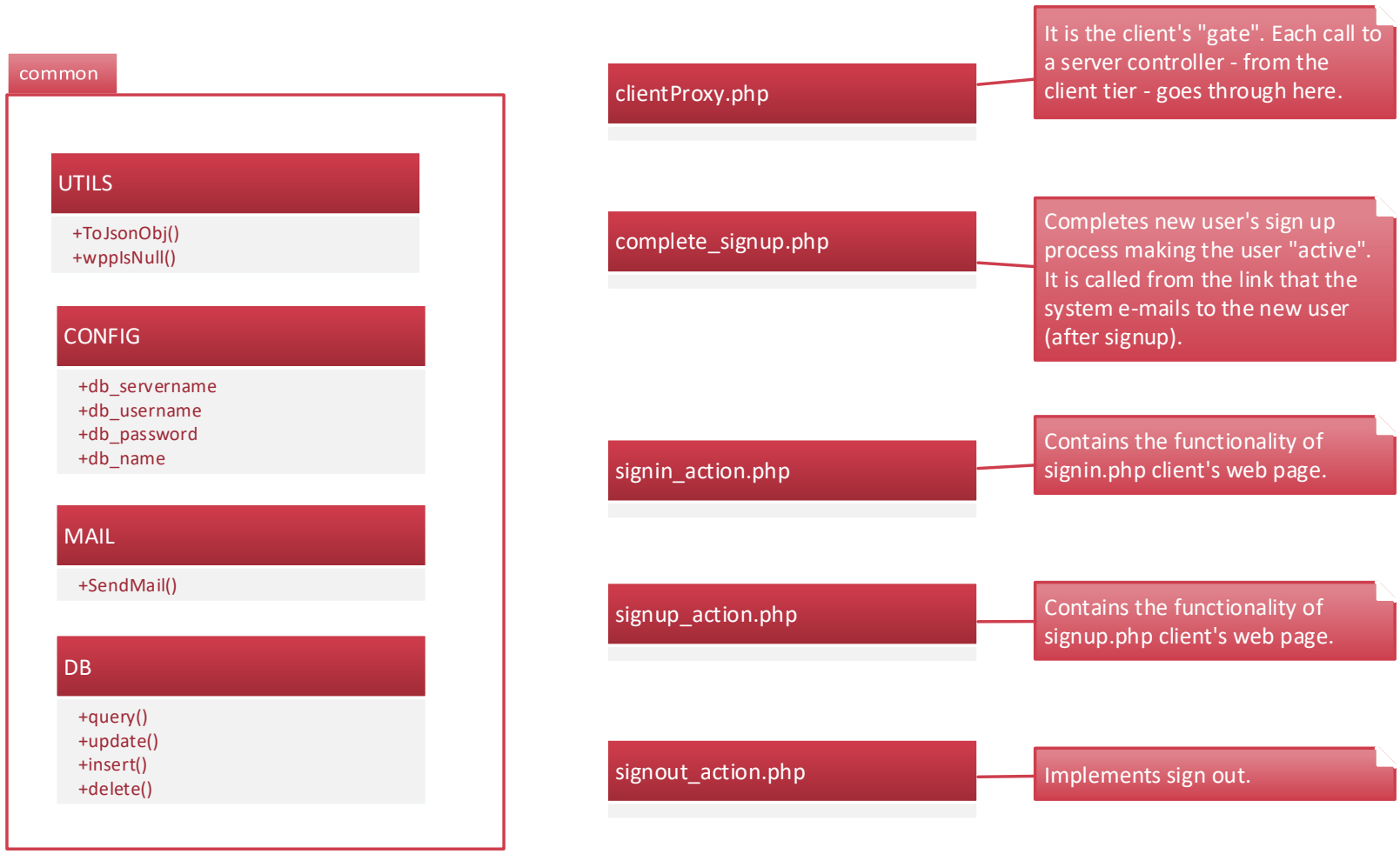


Picture 5: MECIS Database ERD diagram

MECIS Application Server

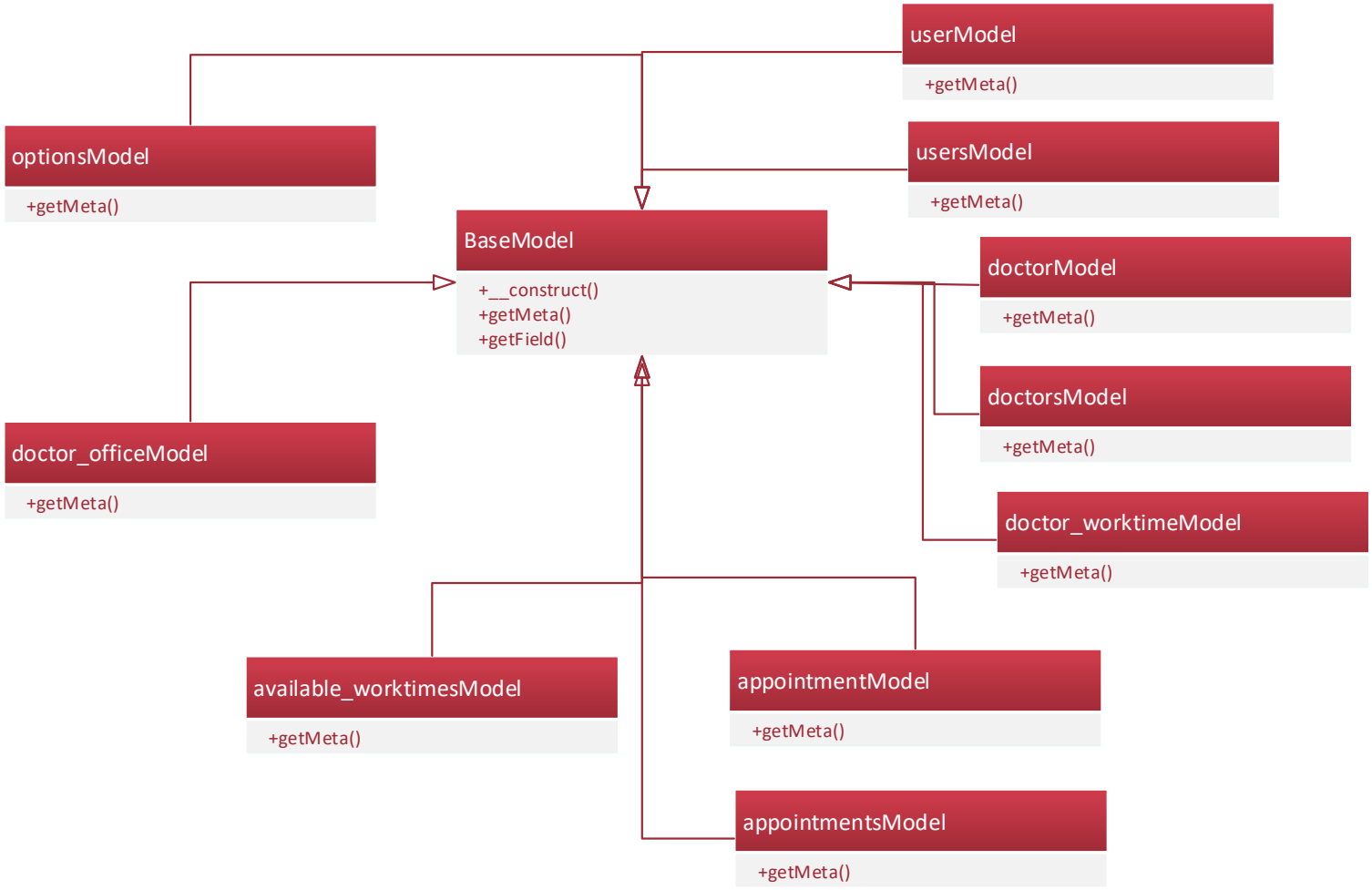
The MECIS Application Server contains mainly the application logic of the system. In addition, it provides all the necessary tools and processes for the database management.

Common subsystem



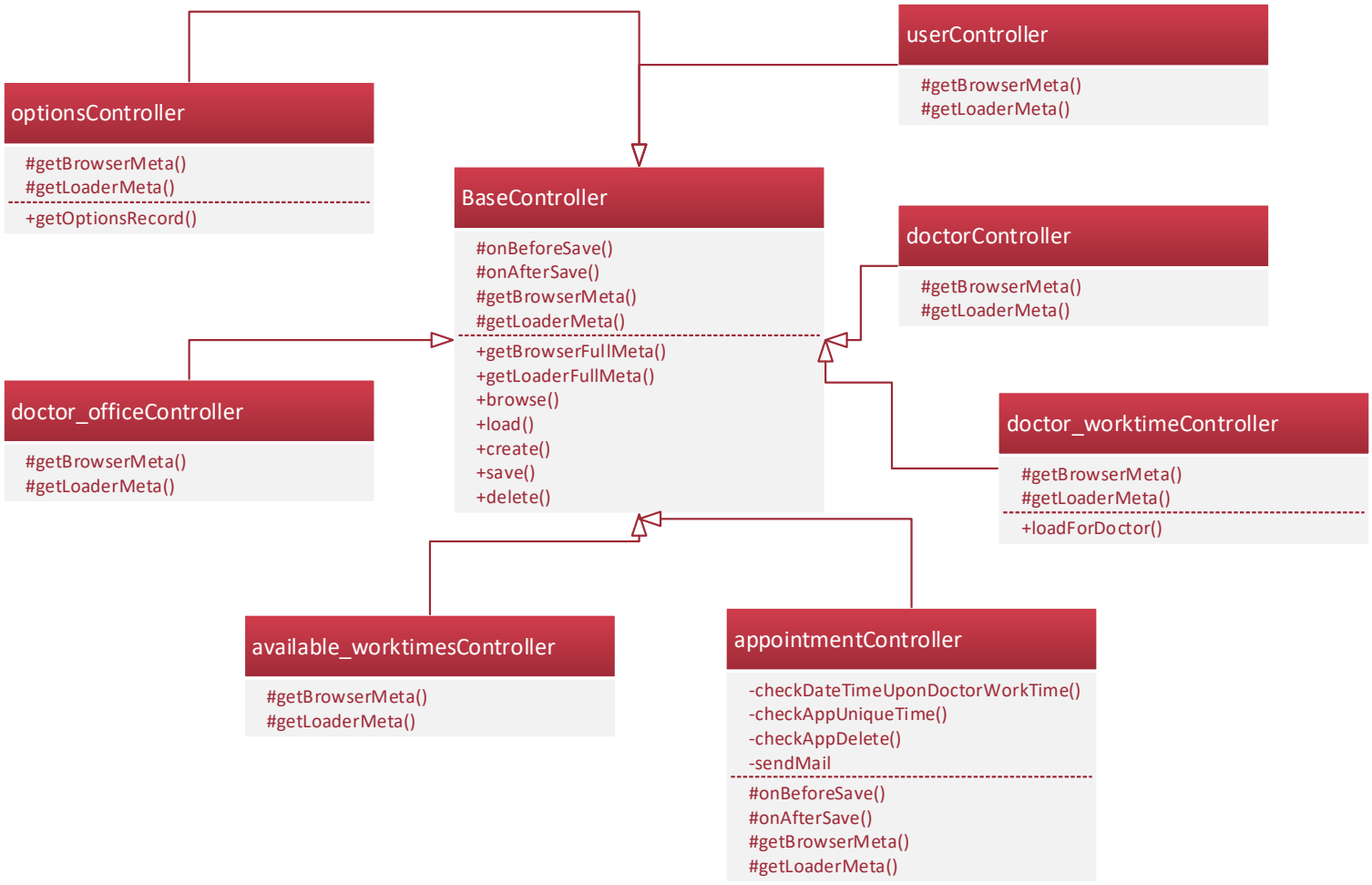
Picture 6: Class diagram for Common Subsystem

Models subsystem



Picture 7: Class diagram for Models Subsystem

***Controllers* subsystem**

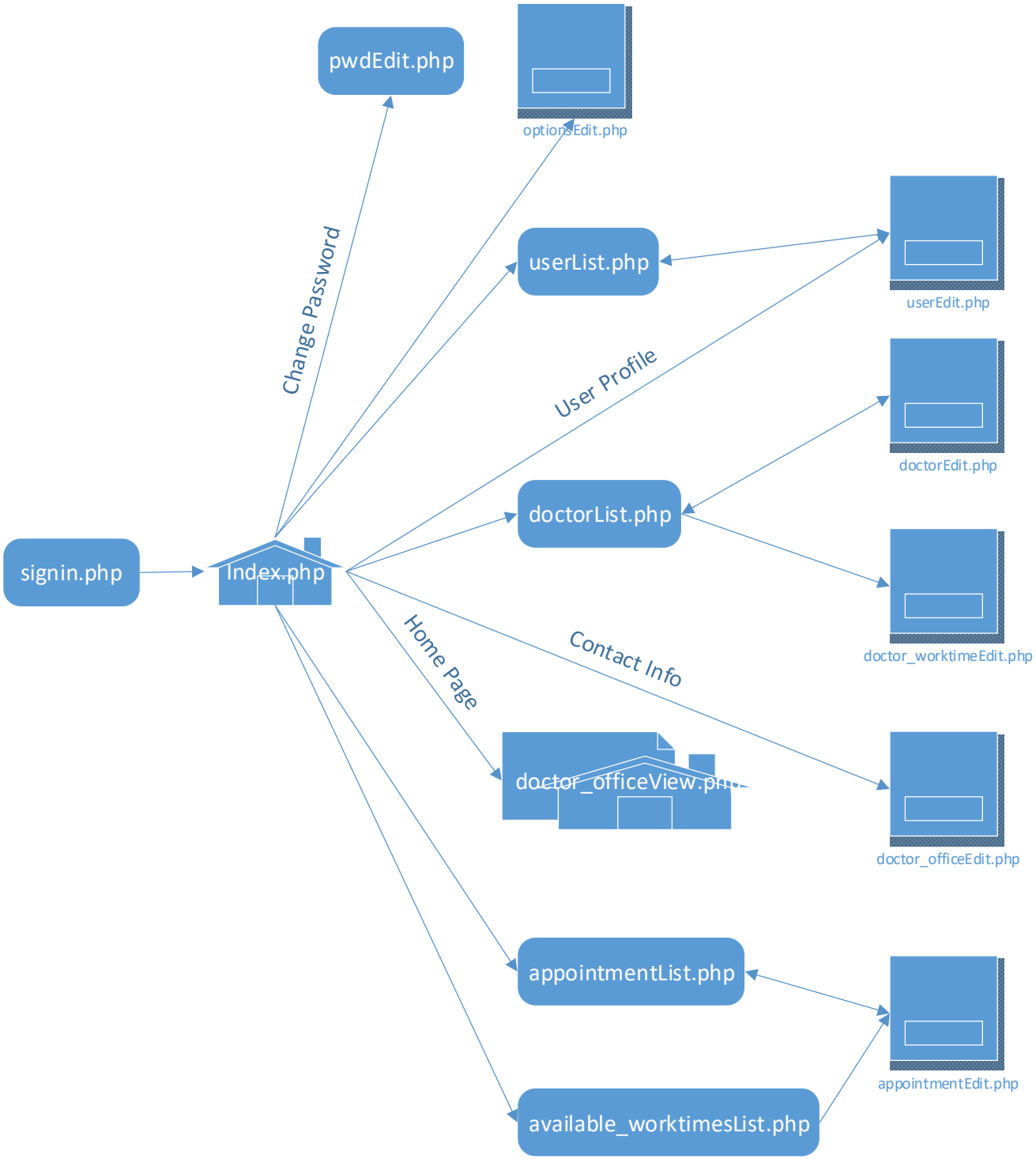


Picture 8: Class diagram for Controllers Subsystem

MECIS Client

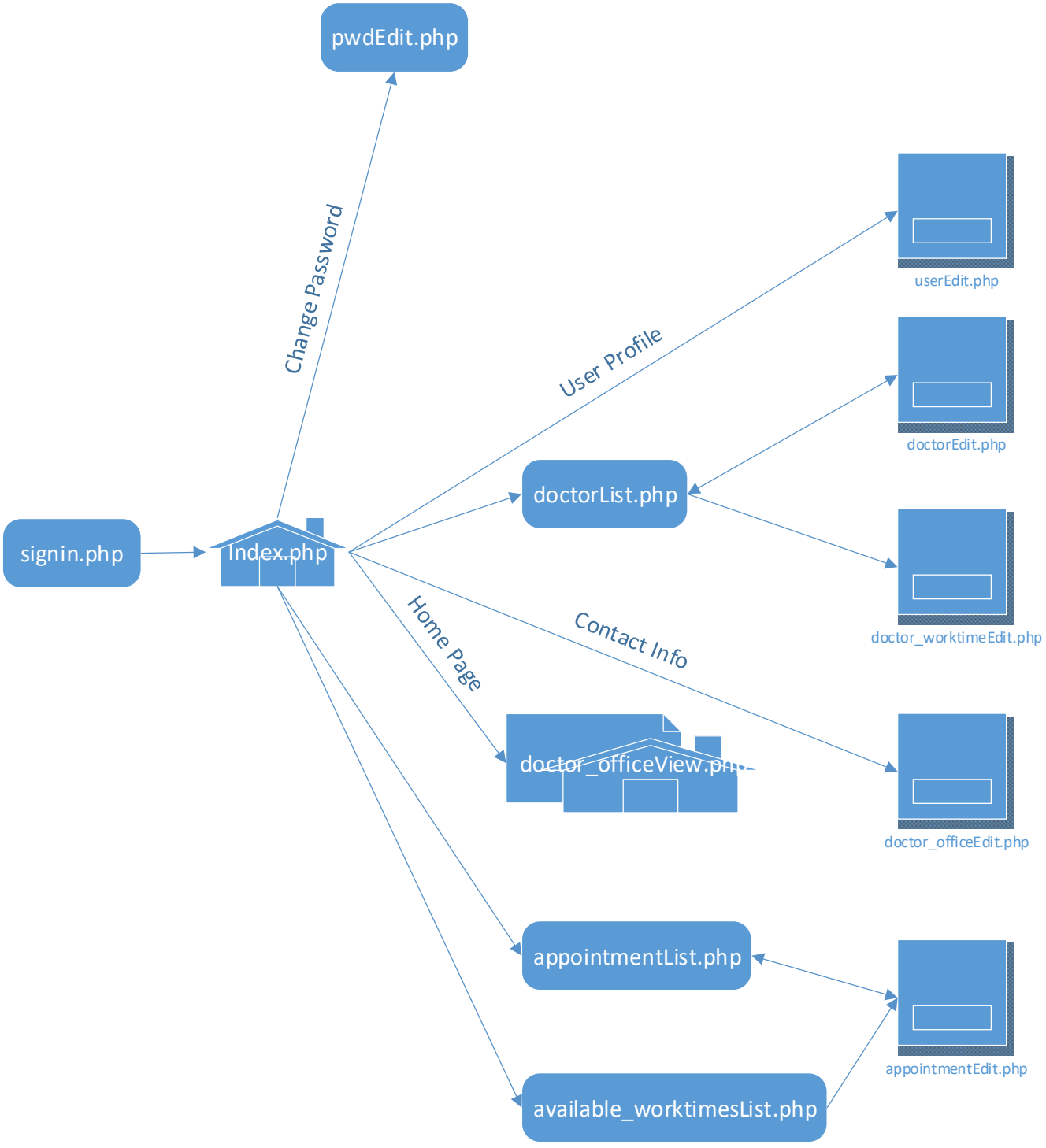
The MECIS Client contains all the UI elements that allow users to communicate with the system. It actually consists of a large number of web pages. Because some of these are only available for specific user roles, the diagrams below present the client web pages per user role separately.

Client perspective for Administrators



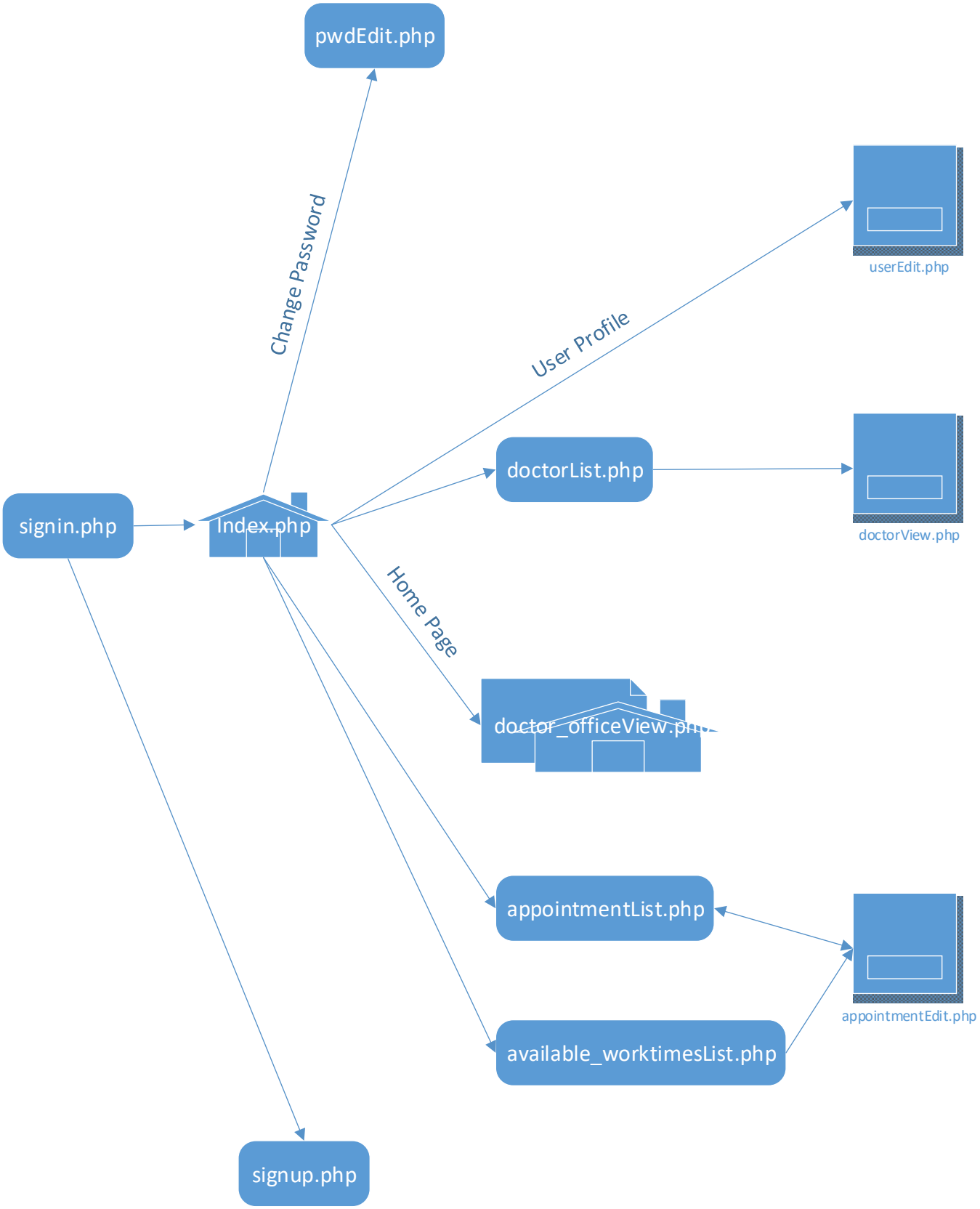
Picture 9: Conceptual web site diagram for Administrators

Client perspective for Secretaries



Picture 10: Conceptual web site diagram for Secretaries

Client perspective for Patients

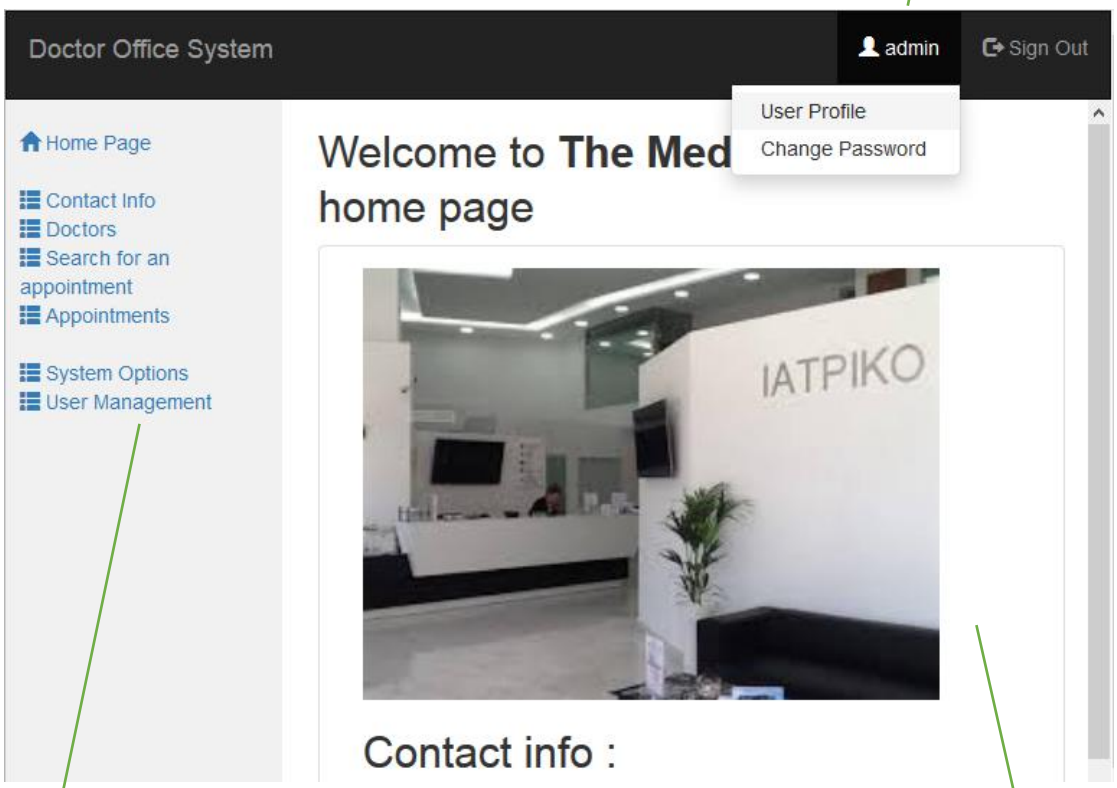


Picture 11: Conceptual web site diagram for Patients

Web pages

Main Page

Current – signed in – user's name. At the same time, acts as a popup menu that offers "User Profile" management and the ability to "Change Password".



Menu area. The menu is dynamic and it is adapted to the role of the current user (that is, it has different content for an administrator than a secretary).

Page area. Each page opens in this area (which contains an iframe).

Browsers

Refreshes the list of records (fetching from the DB the records that satisfy the user's search criteria).

Opens the editor for editing the selected record.

Opens the editor for inserting a new record.

Deletes the selected record (asking for user confirmation first).

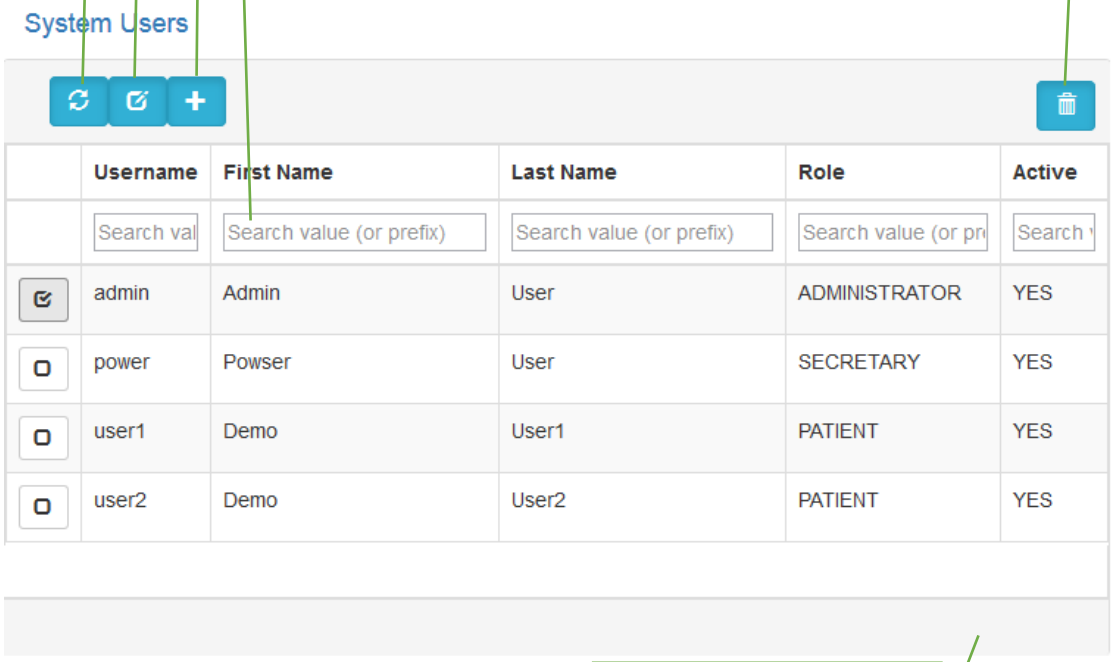
A value that forms a "field search criterion":

field = ***value***

or

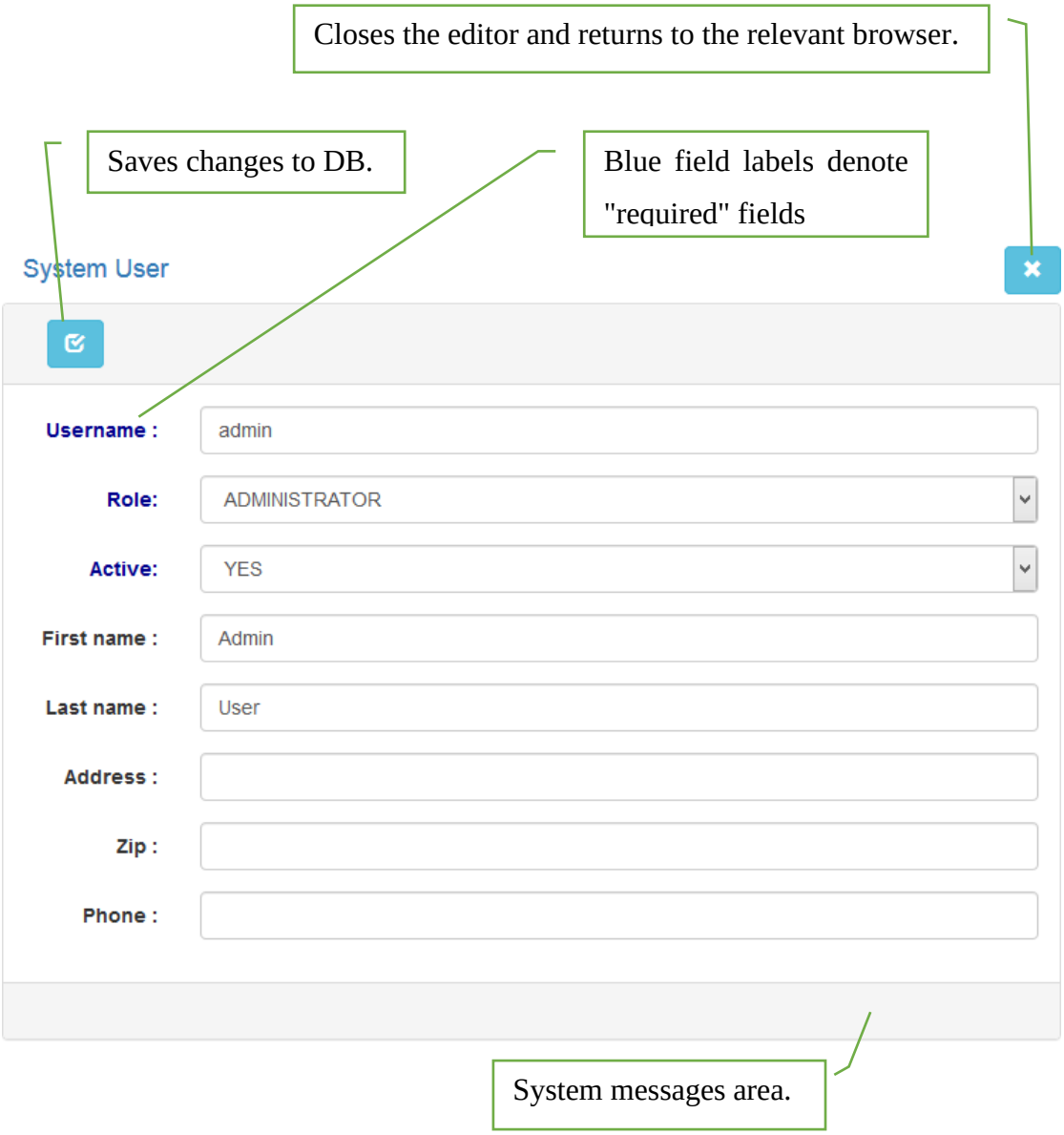
field `like` ***value***`%` (for varchar fields).

All "field search criteria" are combined together with the **and** operator.



System messages area.

Editors



3.3.8 Implementation

MECIS has been developed as a 3-tier web-based application hosted on an Apache Server.

The client tier has been built on HTML, CSS and JavaScript. The application tier has been built on PHP. Finally, the database tier has the form of a MySQL database.

3.3.9 Deployment

In general, the deployment of MECIS follows the classic deployment process of a web application supported by an SQL database.

Prerequisites

The following prerequisites must have already been installed on the deployment machine:

- ✓ An Apache HTTP Server with PHP support (version 5.5 or later)

  When configuring PHP, care must be taken to load "pdo_mysql" extension and have "short_open_tag" option enabled.

- ✓ A MySQL (or MariaDB) Database Server

Deployment process

The deployment material of MECIS consists only of the "**mecis**" folder contained in the "**SourceFiles**" folder that accompanies this thesis document.

So, for the deployment of MECIS, the following process should be followed:

1. Initially, the "**mecis**" folder must be copied into the DocumentRoot folder that has been defined in the Apache configuration file.
2. Using a simple text editor, the credentials of a valid MySQL user (i.e. the "**db_username**" and the "**db_password**") are assigned to the corresponding parameters of the "**mecis\server\common\CONFIG.php**" file.
3. Finally, using a MySQL management tool (e.g. the MariaDB command-line Monitor), the script file "**mecis\wpp.sql**" must be executed in order for the database "**wd_dofa**" to be created.

After completing the above process, the MECIS application can be accessed - from the deployment machine itself - using this URL: **http://localhost/mecis**.

It should also be noted that the MECIS database will initially contain some demo data. These demo data are the following:

Demo Users

Username	Password	User Role
admin	admin	Administrator
power	power	Secretary
user1	user1	Patient
user2	user2	Patient

Demo Doctors

Last Name	First Name	Specialty
Smith	Jim	pathologist
Jones	Tom	pathologist
Thomson	Maria	dermatologist

3.4 MECIS-Bot Planning

As already mentioned, MECIS-Bot plans to be a virtual assistant for the users of the ME-CIS application. Its basic design principles and the requirements it has to address are described below.

3.4.1 Scope

The main scope of MECIS-Bot is to be the key tool for achieving the overall goals of this thesis (see Section 3.2). Thus, MECIS-Bot should be designed and constructed in such a way as to provide the following basic features:

MB-FT1. Provision of interactive user guidance

MB-FT2. Modeling and performing business processes

MB-FT3. Provision of personalized services to the users (i.e. notifications, hints and personal info)

3.4.2 Stakeholders

From the perspective of MECIS-Bot the main stakeholders are:

MB-S1. Employees

All the employees of the medical center that have access to MECIS.

MB-S2. Patients

All the patients of the medical center.

3.4.3 Problems

Like any software, MECIS-Bot is going to help stakeholders address their problems. The problems identified and selected, in this thesis, to be addressed by MECIS-Bot are as follows.

Employees

Employees are presumably advanced users. Therefore, they only need some guidance to perform tasks.

MB-P1. Finding the right workspaces and using the right tools to perform tasks
Employees need guidance to move to appropriate workplaces, within the wider application environment, and use the appropriate tools to perform specific tasks.

Patients

Patients are considered novice users. Therefore, they would appreciate a more human-friendly way of executing tasks, as opposed to using GUI elements.
In addition, they expect some useful personal services (such as various notifications).

MB-P2. Looking for doctors (of a specific specialty or not)

MB-P3. Appointment making

MB-P4. Expending, when making an appointment, the most favorite doctor (by specialty) to proposed as a first choice

MB-P5. Appointment cancellation

MB-P6. Overview of pending appointments

MB-P7. Overview of appointment history

MB-P8. Proactive notification about pending appointments

3.4.4 Constraints

The challenges that a virtual assistant has to face (as presented in Section 2.6.2) impose some basic constraints on MECIS-Bot:

MB-C1. The communication between the user and MECIS-Bot should be as human-friendly as possible and should look like a human conversation.

MB-C2. The conversation between the user and MECIS-Bot should take the form of a series of *stories* [77].

Each *story* has a specific subject, starts with a new user demand and ends with satisfying it. Therefore, during a story, the chatbot tries to "understand" the context of the conversation and lead the user to the expected result.

MB-C3. In the case where the dialogue is deadlocked (according to the user's opinion), the user has the opportunity to start a new story at any time.

MB-C4. The chatbot should offer – to the user - an immediate ability to communicate with a human by various means (e.g. telephone, email, etc.).

3.4.5 Goals

In general, MECIS-Bot intends to offer a solution to the problems presented in Section 3.4.3.

More specifically, the goals to be achieved by MECIS-Bot are different for each stakeholder and are the following:

Employees

MB-G1. Provide direct help to execute a specific task visiting the appropriate workplaces - within the application's wider working environment

Patients

MB-G2. Presentation of the doctors of the medical center (of a specific specialty or not)

MB-G3. Easy and friendly way to make an appointment

MB-G4. Proposal of the favorite doctor (per specialty) based on the history of the appointments, when appointment making

MB-G5. Easy and friendly way to cancel an appointment

MB-G6. Viewing of pending appointments and total appointment history

MB-G7. Automatically show pending appointments notification

3.4.6 System Users

From the perspective of MECIS-Bot, users of the MECIS system are divided into two categories-roles:

MB-U1. Employee

MB-U2. Patient

Description and responsibilities

User Role	Description
Employee	Represents all the MECIS users that are, actually, employees of the Medical Center
Patient	Represents all the patients

User working environment

The user working environment - provided by MECIS-Bot - is common to all users and is immediately accessible, inside the broader MECIS working environment.

3.4.7 System Functional Requirements

The functional requirements that MECIS-Bot must meet derive from the list of problems and constraints MECIS-Bot has to address (as described in sections 3.4.3 and 3.4.4, accordingly).

Employees

MB-FR1. Answering a "how-to" question

The chatbot may guide an employee to do something - inside the application's working environment – by interacting with him (or her) in an interactive way.

Patients

MB-FR2. Displaying doctors

The chatbot can display the doctor list (of a specific specialty or not).

MB-FR3. Making an Appointment

The chatbot can offer the current patient the ability to make an appointment. At the same time, it can suggest the most favorite doctor as the first choice.

MB-FR4. Overviewing of appointments

The chatbot can offer the current patient the ability to display his (or her) appointments (only the pending ones or all of them).

MB-FR5. Canceling an Appointment

The chatbot can offer the current patient the ability to cancel a pending appointment.

MB-FR6. Notifying about pending appointments

The chatbot can automatically notify the current patient about pending appointments.

General requirements

MB-FR7. Restarting the dialogue

The user can restart the dialogue (i.e. terminate the current running story) at any time.

- *After a story termination, the system automatically starts a new story (i.e. accept a new user request).*

3.4.8 System Non-Functional Requirements

Below, are the non-functional requirements that MECIS-Bot should meet.

MB-NFR1. Responding to unclear answers

The system must respond to vague user answers by asking for more clear and specific data.

MB-NFR2. Responding to incomprehensible requests

The system must respond to incomprehensible requests by asking the user to rephrase his (or her) query.

4 Contribution/Experiments

This chapter discusses the MECIS-Bot system, describes its implementation process, and, finally, gives some indicative scenarios of its operation.

4.1 MECIS-Bot System Analysis

In general, the MECIS-Bot system should satisfy the functional and non-functional requirements presented in Sections 3.4.7 and 3.4.8, accordingly. In addition, according to MECIS-Bot planning (see Section 3.4.7), MECIS-Bot should provide largely different functionality for each user role. However, there are some common functions that apply to all users equally.

The following UML use cases outline this system and identify the key elements of its structure and processes. Note that, in all of them, MECIS-Bot is simply referred to as "system".

4.1.1 System Use cases for Employees

[UC11] Making a typical “how-to” question

Title	An Employee makes a “how-to” question
Module	MECIS-Bot
Prime actor	Employee
Other actors	
Startup event	The Employee wants to know how and where to execute a task
Preconditions	The Employee has successfully signed in and the system prompts - the Employee - to make a question
Postconditions	
Basic flow	1. The Employee types his (or her) question (e.g. "*how to deactivate a user?*"). 2. The system starts itself a new story-conversation. 3. If the system has all the data needed to respond, it displays the appropriate information and terminates the story-conversation.
Alternative flows	**The system needs additional data (starts from Step 3 of the basic flow)** 3. The system responds by asking for additional data. 4. The Employee types the additional data. 5. If the system has all the data needed to respond, it displays the appropriate information and terminates the story-conversation. Otherwise, it goes to Step No. 3 of the current alternative flow.

4.1.2 System Use cases for Patients

[UC21] Looking for doctors

Title	A Patient is looking for doctors
Module	MECIS-Bot
Prime actor	Patient
Other actors	
Startup event	The Patient wants to get a list of doctors
Preconditions	The Patient has successfully signed in and the system prompts - the Patient - to make a request
Postconditions	
Basic flow	1. The Patient types an appropriate request (e.g. "*show doctors*"). 2. The system asks for the Patient to select a specific specialty. 3. The Patient selects a specialty. 4. The system displays a list with the doctors of the selected specialty.
Alternative flows	**The Patient wants to see all the doctors regardless of their specialty (starts from Step 3 of the basic flow)** 3. The Patient does not select a specific specialty. 4. The system displays a list of all doctors. **There are no doctors of the selected specialty (starts from Step 4 of the basic flow)** 4. The system displays an appropriate message (e.g. "*There are no Doctors!*").

[UC22] Making an appointment

Title	A Patient is making an appointment
Module	MECIS-Bot
Prime actor	Patient
Other actors	
Startup event	The Patient wants to make an appointment
Preconditions	The Patient has successfully signed in and the system prompts - the Patient - to make a request
Postconditions	
Basic flow	1. The Patient types an appropriate request (e.g. "*make appointment*"). 2. The system displays a unique list of all the specialties of the available doctors. 3. The Patient selects a specialty. 4. The system displays a list of the doctors of the selected specialty (because of the way the system responded to the second step, it is certain that the doctors appearing have some availability). 5. The Patient selects a doctor. 6. The system displays a list of the days on which the selected doctor is available (because of the way the system responded to the second step, it is certain that the list can not be empty). 7. The Patient selects a day. 8. The system displays a list of times on which the selected doctor is available on the selected day (because of the way the system responded to the second step, it is certain that the list can not be empty). 9. The Patient selects a time. 10. The system creates the new appointment and displays an appropriate message.

Alternative flows	**There are no available doctors at all (starts from Step 2 of the basic flow)** 2. The system displays a message like "*There are no available doctors*". **The Patient's request message includes a straightforward reference to a specific specialty (e.g. "*I need a pathologist*" (starts from Step 1 of the basic flow)** The system continues the execution flow from Step 4 of the basic flow. **The Patient does not want to select a specific doctor (starts from Step 5 of the basic flow)** 5. The Patient selects "ANY" instead of selecting a specific doctor. 6. The system displays a list of the days on which at least one doctor, of the selected specialty, is available (because of the way the system responded to the second step, it is certain that the list can not be empty). 7. The Patient selects a day. 8. The system displays a list of all possible combinations between available doctors, of the selected specialty, and the times, on the selected day, they are available (because of the way the system responded to the second step, it is certain that the list can not be empty). 9. The Patient selects a doctor-time pair. 10. The system creates the new appointment and displays an appropriate message.

	The system identifies that the most Patient's favorite doctor, of the selected specialty, is available (starts from Step 4 of the basic flow) 4. The system recommends the favorite doctor to the Patient. 5. The Patient accepts the proposal. 6. The system displays a list of the days on which the doctor is available (because of the way the system responded to the second step, it is certain that the list can not be empty). 7. The Patient selects a day. 8. The system displays a list of times on which the doctor is available on the selected day (because of the way the system responded to the second step, it is certain that the list can not be empty). 9. The Patient selects a time. 10. The system creates the new appointment and displays an appropriate message.

[UC23] Notifying about pending appointments

Title	The system notifies the current Patient about pending appointments
Module	MECIS-Bot
Prime actor	
Other actors	
Startup event	The Patient signs in
Preconditions	The Patient has successfully signed in and there are pending appointments of him (or her)
Postconditions	
Basic flow	1. The system displays a notification containing a list with all pending appointments.
Alternative flows	

[UC24] Showing appointments

Title	A Patient asks for showing the appointment list
Module	MECIS-Bot
Prime actor	Patient
Other actors	
Startup event	The Patient wants to get his/her appointment list
Preconditions	The Patient has successfully signed in and the system prompts - the Patient - to make a request
Postconditions	
Basic flow	1. The Patient types an appropriate request (e.g. "*show appointments*"). 2. The system displays a list with all the appointments.
Alternative flows	**The Patient wants to see only the pending appointments (starts from Step 1 of the basic flow)** 1. The Patient clarifies, at his/her request, that he/she is only interested in pending appointments (i.e. typing "*show pending appointments*"). 2. The system displays a list with only pending appointments. **There are no appointments (starts from Step 2 of the basic flow)** 2. The system displays an appropriate message (e.g. "*There are no Appointments!*").

[UC25] Canceling an appointment

Title	A Patient cancels an appointment
Module	MECIS-Bot
Prime actor	Patient
Other actors	
Startup event	The Patient wants to cancel an appointment
Preconditions	The Patient has successfully signed in and the system prompts - the Patient - to make a request
Postconditions	
Basic flow	1. The Patient types an appropriate request (e.g. "*cancel appointment*"). 2. The system displays a list with all the pending appointments. 3. The Patient selects an appointment. 4. The system cancels the appointment and displays an appropriate message.
Alternative flows	**There are no pending appointments (starts from Step 2 of the basic flow)** 2. The system displays an appropriate message (e.g. "*There are no pending Appointments!*").

4.1.3 General System Use Cases

[UC31] Restarting the dialogue

Title	A patient asks for restarting the dialogue
Module	MECIS-Bot
Prime actor	User (Employee or Patient)
Other actors	
Startup event	The user wants to restart the dialogue – with the bot – to make a new request
Preconditions	The user has successfully signed in and developed a dialogue with the system
Postconditions	The system prompts - the user - to make a new request
Basic flow	1. The user presses the "Restart" button. 2. The system terminates the current story-conversation.
Alternative flows	

4.2 Development Technologies, Tools and Languages

There are some technologies that play a fundamental role in the modeling and implementation of MECIS-Bot, even in the early stages of these processes. These technologies are presented below.

4.2.1 Rasa Framework

Rasa [78] is a machine learning framework for automated text and voice conversations. In fact, it provides the entire necessary infrastructure for developing contextual AI assistants and chatbots. In other words, it is a framework. It is open source and free but that does not prevent it from competing well with all known non-free frameworks in the chatbot area [79].

Rasa can understand messages, develop conversations and connect to messaging channels and APIs. It is also customizable and it can be intergraded into any existing system.

Architecture

Rasa is mainly consisting of two modules:

- **Rasa NLU**
- **Rasa Core**

Rasa NLU

Rasa NLU [80] is an open-source natural language processing tool for *intent* recognition and *entity* extraction, in chatbots. For example, if you get a sentence like "*I'm looking for an Italian restaurant in Athens*" it will return something like the following structure:

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine" : "Italian",
    "location" : "Athens"
  }
}
```

Rasa core

Rasa core [81] is a dialogue engine for AI software. It is a key part of the Rasa framework. It is mostly built in the Python programming language and is, generally, based on open standards and open software.

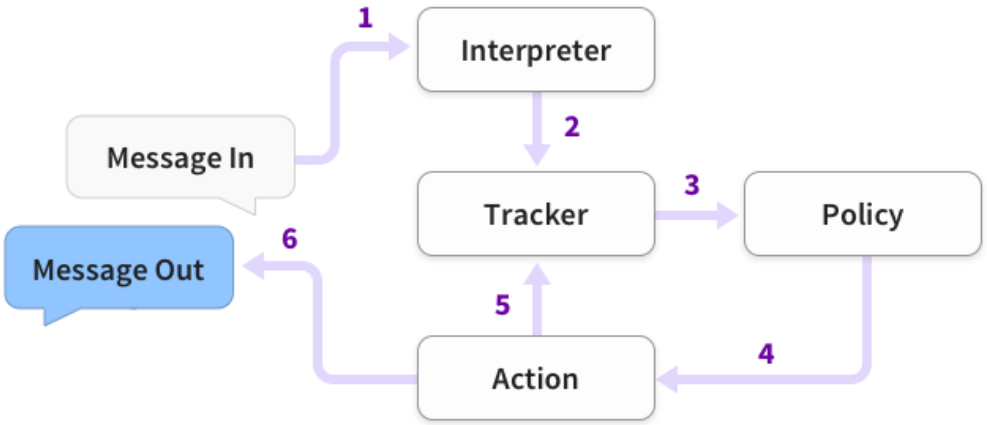
The key element of its philosophy is that the next step - in a dialogue - is not based on some kind of programming code but on properly trained machine learning models. This approach [82] enables the manufacturer to create chatbots that can hold a contextual conversation without having to code all possible variants of the conversation.

Conversation management

The dialogue between the user and Rasa develops as a series of stories. According to Rasa, a *story* is a conversation between a user and a chatbot that is characterized by a specific context and is identified through user messages and chatbot responses.

Message handling

According to Rasa architecture [83], the typical flow of the handling process of a received message is shown in the following diagram of the Rasa documentation.



Picture 12: Rasa message handling process

In particular, the above process consists of the following steps:

1. The received message is transmitted to an *interpreter*, who recognizes the intention expressed by the message and extracts any entities contained therein.

   The "Interpreter" is handled by Rasa NLU.

2. Then, the Interpreter sends the extracted information to an object called "Tracker" which is, actually, responsible for keeping track the state of the whole conversation.

   The Tracker and all other objects involved in the next steps are handled by Rasa Core.

3. The Tracker sends the above information, along with the current state of the conversation, to the "Policy" object.
4. The Policy chooses which *action* to take next and asks for it to be executed.
5. The selected "Action" generates a return message, based on its definition, and informs the Tracker of its execution.
6. Finally, the Rasa Core returns the generated message to the user.

Communication channels

There are indeed many ways for a software to communicate with Rasa. The most common of these are presented below.

Python libraries

It is to be expected that, since Rasa is built with Python, this is the most obvious method – for an application - to communicate with Rasa NLU and Rasa Core, too. Unfortunately, however, this requires code development in Python, which is not necessarily easy or desirable.

HTTP API

The Rasa Framework provides (in fact, of course, this is an offer of the last versions of Rasa) a complete open REST API based on HTTP. This API offers endpoints to manage any phase of the conversation management.

Certainly, this is a method of communication that does not require knowledge of Rasa's internal structures and functions. Additionally, it does not require code development in Python.

Rasa servers

Each Rasa installation has two special servers. Actually, these servers represent Rasa for any other application needs to provide chatbot services to its users. Therefore, the only thing the application has to do, to exploit these servers, is to use the above HTTP API.

Rasa (main) server

It is a HTTP server handling requests, receiving through a rich API, based on Rasa Core. It is, actually, the main server of a Rasa installation.

Rasa actions server

It is also an HTTP Server, but it is used exclusively by Rasa Core. Its only role is to perform specialized advanced actions for the sake of Rasa Core.

Rasa project

A *Rasa project* contains all the components required for a Rasa installation to operate. These components are stored inside specific files following a specific format.

Initially, a Rasa project is created using the tools that the Rasa Framework provides. The chatbot developer is, then, responsible to create the appropriate objects, inside the project files.

The main categories of objects of a Rasa project are described below.

Intents

An *intent* represents the intention that is hidden within a user message. Practically, it is a keyword that describes the meaning of the whole phrase.

According to Rasa, all intents that need to be understood - by the chatbot - during the various conversations, must be recorded in a file called "nlu.md". For example, a list of intents, inside this file, looks like below:

```
## intent:check_balance
- what is my balance

## intent:greet
- hey
- hello
```

The intents must also be recorded in the "domain.yml" file, as shown below:

```
intents:
  - check_balance
  - greet
```

Entities

Entities represent the pieces of information that can be extracted by user messages. For example, from the "*show me chinese restaurants*" phrase, the "cuisine" entity can be extracted with the "chinese" value.

According to Rasa, all entities that require special processing must be recorded in the "domain.yml" file. For example, a list of entities looks like below:

```
entities:
 - cuisine
 - name
```

Actions

Actions are the things that the chatbot runs in response to user input. There are many kinds of actions in Rasa:

- **Utterance actions**

  Start with **utter_** and send a specific message to the user.

- **Custom actions**

  They are defined by the chatbot developer, writing Python code, and can perform various tasks. Finally, they usually return one or more messages to the user.

- **Default actions**

  They are defined by Rasa itself and have a specific impact (e.g. conversation re-start etc.).

According to Rasa, all actions except Default must be recorded. In addition, each Utterance action must be defined by an appropriate utterance template (containing, actually, the message to be displayed by the chatbot). Finally, the code that defines each Custom action must be written in Python inside a python script file (which, by convention, is called "actions.py").

For example, a list of Utterance actions along with the corresponding templates looks like below:

```
actions:
- utter_greet
- utter_happy

templates:
  utter_greet:
  - text: "Hey! How are you?"

  utter_happy:
  - text: "Great, carry on!"
```

Forms

One of the most common parts of a human-chatbot conversation is information gathering. In these cases, the chatbot needs a series of data to perform an action (making appointments, etc.).

A Rasa *form* is a special tool that is capable of gathering information from the user in a specific order. At the same has the ability to validate the inserted data.

In fact, Rasa forms are custom actions created by the chatbot developer writing code in the Python language. Therefore, in a Rasa project, all forms are located in the "actions.py" file.

Stories

Rasa stories are conversation examples that are used to "train" the Rasa Core. In fact, they define the conversation cases that the chatbot can manage when communicating with the user.

All stories must be recorded in a file called "stories.md". A story example follows below:

```
## greet + location/price + cuisine + number of people
* greet
   - action_ask_howcanihelp
* inform{"location": "athens", "price": "very cheap"}
   - action_on_it
   - action_askfor_cuisine
* inform{"cuisine": "greek"}
   - action_ask_numberofpeople
* inform{"people": "four"}
   - action_ack_dofind
```

4.2.2 BotUI Framework

BotUI [84] is an open source framework to build UI for chatbots. It is based on JavaScript and Vue [85]. It provides a JavaScript API for displaying a variety of UI controls, inside a specific "client area".

BotUI has nothing to do with the conversation management at all. It is just a framework for visualizing the conversations content.

According to BotUI, the client area of a chatbot is provided by a compound HTML component that is created as an instance of the BotUI class. Consequently, all UI elements that the chatbot wants to display - to the user - are contained within it.

UI elements

The main UI elements that the BotUI framework offers to chatbot developers are as follows:

- **Message**

  A static text inside an ellipse. It can be shown as a chatbot message (with gray background color at the left) or a human message (with blue background color at the right).

- **Text Box**

  A simple text box that allows the user to type some text.

- **Button Group**

  A set of buttons each expressing a different choice. The user can click on only one of them.

- **Single Selection Dropdown List**

  A dropdown list of different options. The user can select only one of them.

4.3 MECIS-Bot System Modeling

MECIS-Bot should be regarded as a subsystem of the MECIS system. Nevertheless, it has its own autonomy and it must meet its own requirements (see Sections 3.4.7 and 3.4.8). In addition, it must provide the appropriate environment for the seamless execution of the use cases of Section 4.1.

It is obvious that, the choice of Rasa as the framework for the development of MECIS-Bot imposes certain assumptions on the system-modeling phase.

At the same time, of course, many key components of MECIS-Bot are based on the Rasa Framework and take advantage of its benefits.

4.3.1 Human-Chatbot Conversation Design

In general, the conversation between MECIS-Bot and the user is based on the concept of *story* which is also the basic concept of Rasa Framework (see Section 4.2.1). According to Rasa, a story is not just a process of exchanging messages but also an entity that has specific characteristics and behavior.

Conversation unit

According to MECIS-Bot, the conversation is evolving as a series of independent stories. Under this approach, a story represents a distinct *conversation unit*.

MECIS-Bot extends further the concept of story and adds a set of specific specifications to it:

- Each story has a specific conceptual context that is automatically identified by the chatbot itself.
- Each story has a beginning and an end.
    - Each story is started by the chatbot with an initial welcome message.
    - Each story ends in two ways:
        - The chatbot "concludes" that the purpose of the conversation has been achieved and, at the same time, presents the expected results.

      Or

        - The user has requested its termination.
- The termination of a story simultaneously triggers the start of a new story.

Conversation block

According to MECIS-Bot, each story is technically composed as a series of *conversation blocks*. In general, a conversation block (see Picture 13) consists of three elements:

- **Chatbot message**

  It is either a prompting message (prompting the user to take some action) or a reply message from the previous conversation block.

  By convention, it always appears on the left, in gray background color.

- **Human's working area**

  It consists of one or more UI input elements that allow the user to respond to the chatbot's prompt by actually creating a message to the chatbot.

  By convention, it always appears under the chatbot message.

- **Human message**

  The actual message that the user created above.

  By convention, it always appears on the right, in blue background color.

Chatbot message

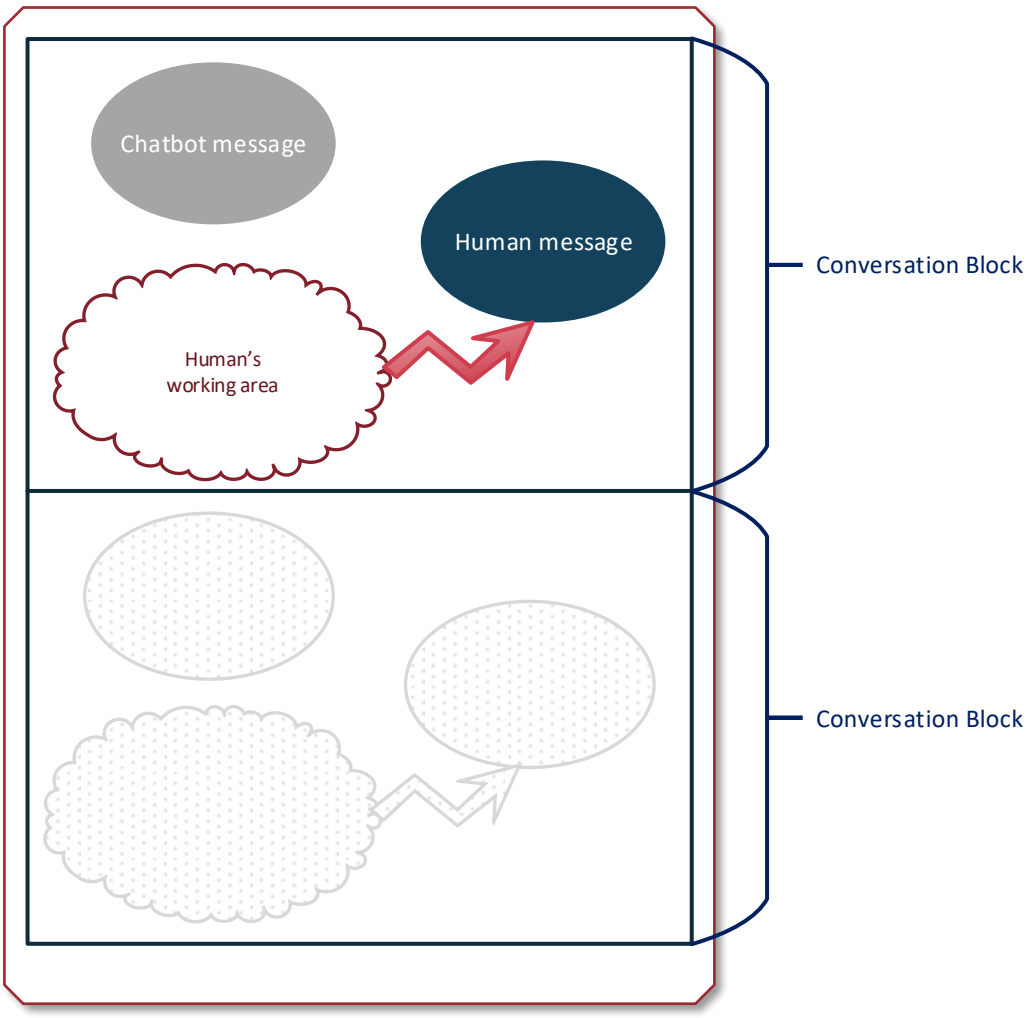
A chatbot message consists of one or more of the following UI elements:

- **Text**
- **Icons**

  Icons make text more readable.

- **Links**

  Html links that lead to a specific URL.



Picture 13: MECIS-Bot conversation block

Human's working area

In chatbots that follow the monolithic design style, the human's working area is fixed and it is located outside the conversation area (see Section 2.5.6).

Unlike many commercial chatbots, MECIS-Bot follows the interactive design style. So, the human's working area is dynamic in terms of both location and content.

In fact, MECIS-Bot displays the human's working area under every prompt message. This allows the user to respond by keeping the chatbot's message in its field of view.

In particular, MECIS-Bot offers to the user - in order to create input - the following input UI elements:

- **Text Boxes**

  A Text Box allows the user to type some text.

- **Button Groups**

  A Button Group is a set of buttons each expressing a different choice. The user can click on only one of them.

- **Single Selection Dropdown Lists**

  A Dropdown List offers a list of unique options to the user. Then, the user selects one of them.

Conversation process

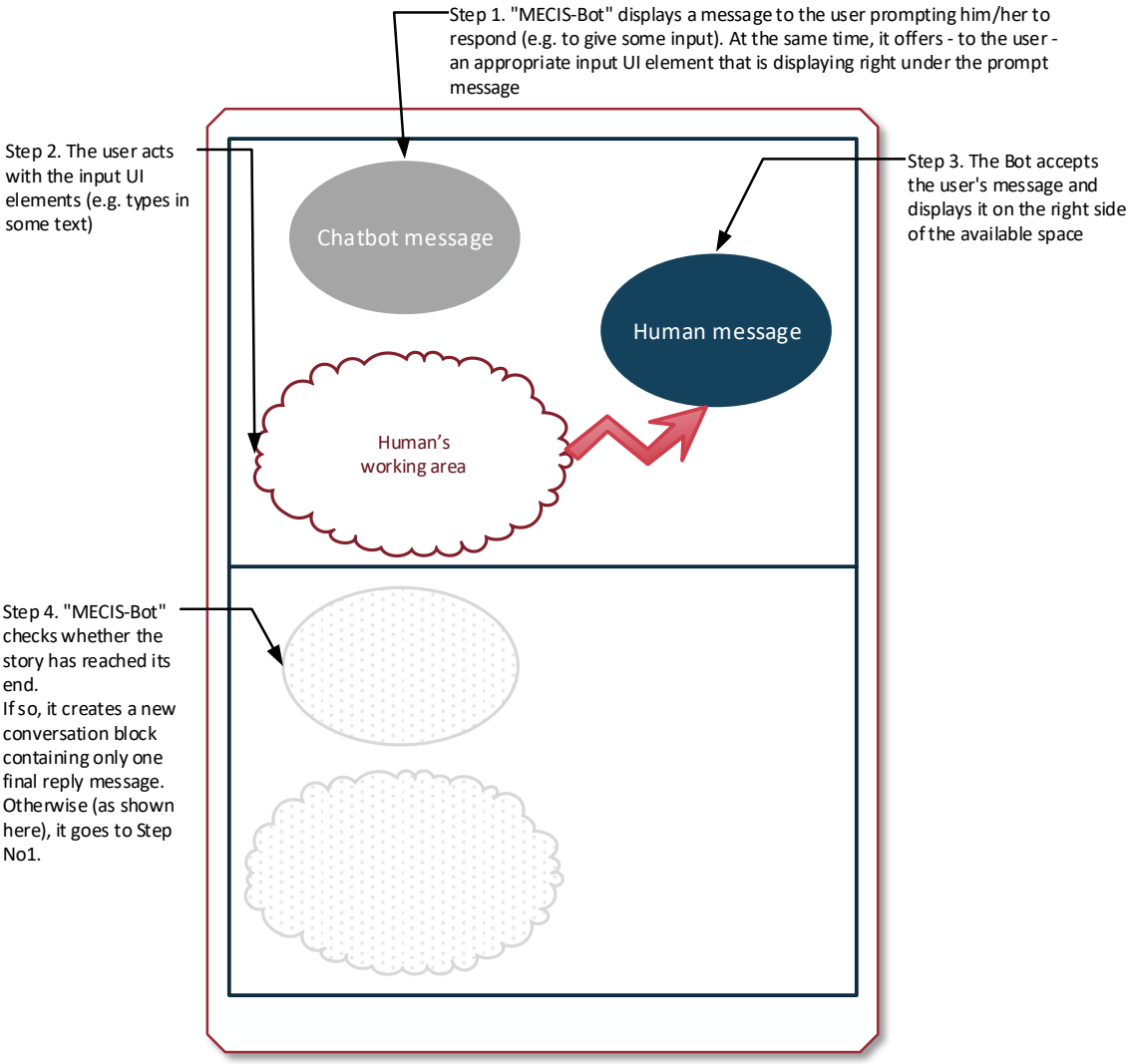
The process, by which a conversation is developed and conversation blocks are created, is a sequence of four steps (see Picture 14):

Step 1. MECIS-Bot creates a new conversation block and displays a message to the user prompting him/her to respond (e.g. to give some input).
At the same time, it offers – to the user - an appropriate input UI element that is displaying right under the prompt message.

Step 2. The user acts with the input UI element and provides some data (e.g. types in some text).

Step 3. MECIS-Bot receives the input data and displays it, as a text message, on the right side of the available space.

Step 4. MECIS-Bot checks whether the story has reached its end.
If so, it creates a new conversation block containing only one final reply message.
Otherwise, it goes to Step No1.



Picture 14: MECIS-Bot conversation process

4.3.2 System Architecture

The architecture of the MECIS-Bot system should be studied in the context of the architecture of the MECIS system itself. In any case, of course, it must meet the above principles of conversation design.

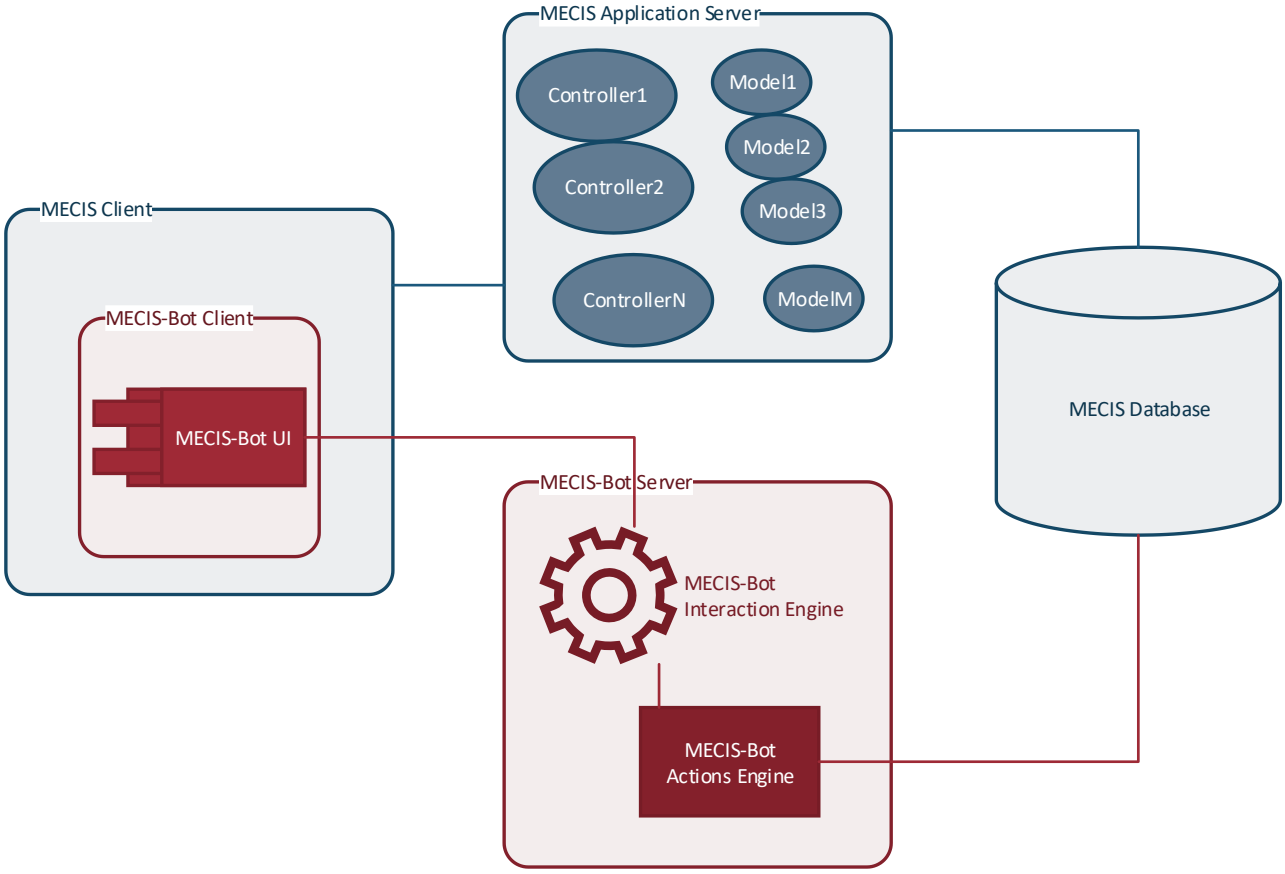
As described in Section 3.3.7, MECIS is a 3-tier application consisting of the following tiers:

- MECIS Database (i.e. Database Server)
- MECIS Application Server
- MECIS Client

So, the components of MECIS-Bot should either be included in some of these tiers or form separate architectural tiers.

More specifically, the main components that make up MECIS-Bot are as follows (see Picture 15) and they are distributed over two architectural tiers (i.e. "MECIS-Bot Client" and "MECIS-Bot Server"):

- **MECIS-Bot UI** } **MECIS-Bot Client**
- **MECIS-Bot Interaction Engine** } **MECIS-Bot Server**
- **MECIS-Bot Actions Engine**



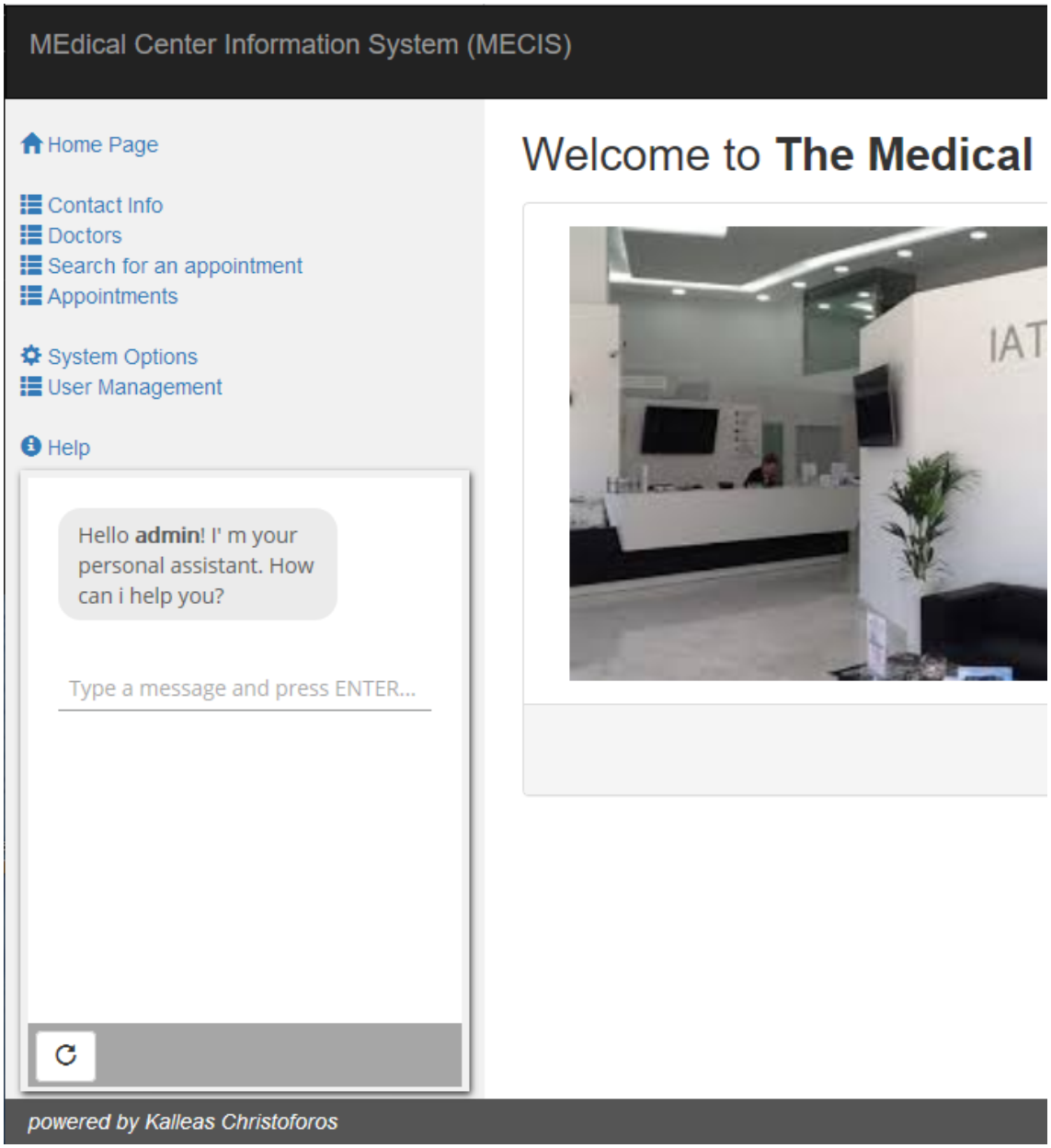
Picture 15: MECIS-Bot system architecture

More details on MECIS-Bot components are given in the following sections.

4.3.3 MECIS-Bot Client

The MECIS-Bot Client provides the MECIS-Bot with the UI required to communicate with all users, regardless of their role (i.e. Employ or Patient). In fact, it works the same way for both Employees and Patients. At the same time, it keeps a link to the Interaction Engine that is, indeed, the "brain" of the system.

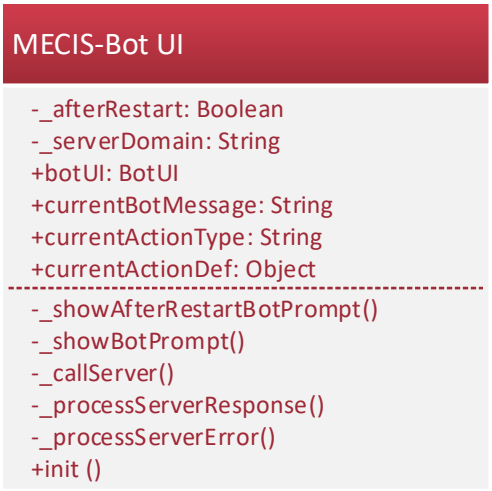
In terms of architecture, the MECIS-Bot Client is fully integrated into the broader MECIS Client as a typical visual component (see Picture 16). However, it retains its autonomy and, based on its design, can be integrated into any other application.



Picture 16: MECIS-Bot Client as a visual component of the MECIS main page

Architecture

MECIS-Bot Client consists of a single UI component called "MECIS-Bot UI". This component implements all the necessary functionality. It is defined by the homonymous class shown in Picture 17 and described below.



Picture 17: MECIS-Bot Client Architecture (Class Diagram)

MECIS-Bot UI

MECIS-Bot UI is a visual component that can be considered as part of the MECIS main page (see Picture 16). Thus, it is automatically activated when the main page is loaded.

Properties

- **botUI**

  The visual component that visualizes the client area of the chatbot.

- **currentActionType**

  The type of the input UI element that the chatbot is going to display. The default value of this property is ***text***, which means that the chatbot is going to display a simple text box.

  As described in Section 4.3.1, the possible options are as follows: Text, Buttons or Options.

- **currentBotMessage**

  The message that the chatbot is going to display.

- **currentActionDef**

  The UI element that the chatbot is going to display.

Methods

- **_showBotPrompt()**

  Displays – inside the client area of the chatbot – either the current message or the current UI element, according to currentActionType.

- **_callServer()**

  Sends a message to the MECIS-Bot Interaction Engine.

- **init()**

  Initializes the chatbot and clears the client area.

4.3.4 MECIS-Bot Server

MECIS-Bot Server contains the key components of the MECIS-Bot. More specifically, these components are the following:

- **MECIS-Bot Interaction Engine**
- **MECIS-Bot Actions Engine**

As can be seen from the system analysis of MECIS-Bot (see Section 4.1), the system offers a completely different set of services to employees than to patients. After all, the conversations with employees have a completely different background from those with patients.

This finding is very crucial for modeling the MECIS-Bot Server. It indicates clearly the need to subdivide, each of the above components, into two object packages (one for Employees and one for Patients).

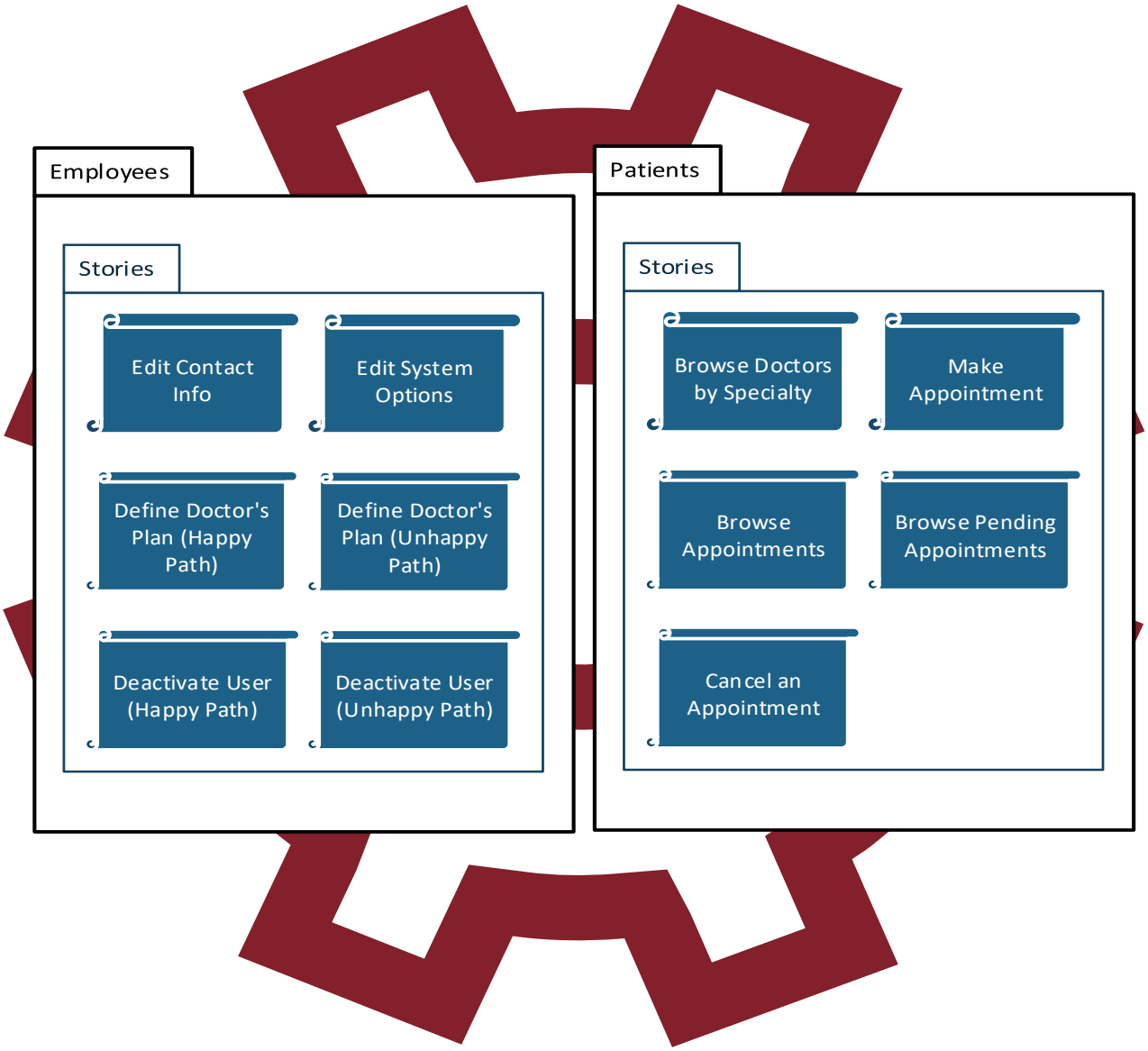
MECIS-Bot Interaction Engine

The MECIS-Bot Interaction Engine represents, actually, the Rasa Core [81]. Therefore, according to the principles of the Rasa Framework, it is responsible for the development of the interactive conversation with the users. To accomplish this, instead of using some kind of programming logic, it relies on a machine-learning model "trained" on example conversations (i.e. the stories).

Architecture

At first, following the general modeling direction of the MECIS-Bot Server, all the objects that make up the MECIS-Bot Interaction Engine are subdivided in two independent packages (i.e. Employees and Patients).

According to Rasa Framework (see Section 4.2.1), the key objects of an Interaction Engine should be the set of stories that the chatbot can develop, when communicating with the user. Thus, the architecture of the MECIS-Bot Interaction Engine is essentially determined by the stories this can execute (see Picture 18).



Picture 18: MECIS-Bot Interaction Engine Architecture

Stories

More specifically, the MECIS-Bot stories are described in the following tables (see Table 2 for Employees and Table 3 for Patients).

Title	Code	Description
Edit Contact Info	STORY-E1	The user-employee is guided to edit the contact info of the Medical Center
Edit System Options	STORY-E2	The user-employee is guided to edit the system options of the MECIS application
Define Doctor's Plan (happy path)	STORY-E3	The user-employee is guided to insert the weekly working schedule of a doctor
Define Doctor's Plan (unhappy path)	STORY-E4	The user-employee is guided to insert the weekly working schedule of a doctor but, for unknown reasons, the doctor does not exist (in the database)
Deactivate User (happy path)	STORY-E5	The user-employee is guided to deactivate a system user
Deactivate User (unhappy path)	STORY-E6	The user-employee is guided to deactivate a system user but for unknown reasons, the user does not exist (in the database)

Table 2: Stories for Employees

Title	Code	Description
Browse Doctors by Specialty	STORY-P1	The user-patient gets - on screen - a list of doctors of a specialty
Make Appointment	STORY-P2	The user-patient makes an appointment
Browse Appointments	STORY-P3	The user-patient gets - on screen - a list of its appointments (pending or not)
Browse Pending Appointments	STORY-P4	The user-patient gets - on screen - a list of its pending appointments
Cancel an Appointment	STORY-P5	The user-patient cancels a pending appointment

Table 3: Stories for Patients

Intents

The startup event for the execution of each of the above stories is, always, a user request (i.e. an intent, according to Rasa terminology) [86]. More specifically, the intents that MECIS-Bot can identify are described in the following tables (see Table 4 for Employees and Table 5 for Patients, below).

Code	Description	Starts …
edit_contact_info	The user-employee is asking how to edit the contact info of the Medical Center	STORY-E1
edit_system_options	The user-employee is asking how to edit the system options	STORY-E2
define_plan	The user-employee is asking how to define a doctor's weekly working schedule	STORY-E3, STORY-E4
deactivate_user	The user-employee is asking how to deactivate a system user	STORY-E5, STORY-E6

Table 4: Intents for Employee stories

Code	Description	Starts…
browse_doctors	The user-patient requests to see the doctors	STORY-P1
make_appointment	The user-patient requests to make an appointment	STORY-P2
browse_appointments	The user-patient requests to see all its appointments (pending or not)	STORY-P3
browse_pending_appointments	The user-patient requests to see all its pending appointments	STORY-P4
cancel_appointment	The user-patient requests to cancel an appointment	STORY-P5

Table 5: Intents for Patient stories

Actions

During the course of each of the above stories, each user input ends up being executed - by Rasa - an action [86]. More specifically, the actions that MECIS-Bot can execute are described in the following tables (see Table 6 for Employees and Table 7 for Patients).

Code	Type	Description
utter_edit_contact_info	Utterance	The chatbot indicates the page where the user-employee can edit the contact info
utter_edit_system_options	Utterance	The chatbot indicates the page where the user-employee can edit the system options
utter_find_doctor	Utterance	The chatbot asks the user-employee to go to the doctor search page
utter_edit_plan	Utterance	The chatbot indicates the tool where the user-employee can specify a doctor's working schedule
utter_doctor_does_not_exist	Utterance	The chatbot displays an error message when the requested doctor does not exist
utter_find_user	Utterance	The chatbot asks the user-employee to go to the user search page
utter_edit_user	Utterance	The chatbot indicates the tool where the user-employee can edit a user's data
utter_user_does_not_exist	Utterance	The chatbot displays an error message when the requested user does not exist

Table 6: Actions for Employee stories

Code	Type	Description
action_browse_doctors	Custom	The chatbot shows – on screen – the doctors of a specialty
action_browse_appointments	Custom	The chatbot shows – on screen – the appointments of the current user-patient
action_browse_pending_ appointments	Custom	The chatbot shows – on screen – the pending appointments of the current user-patient
appointment_form	Form	The chatbot activates the making appointment process
cancel_appointment_form	Form	The chatbot activates the canceling appointment process
utter_ask_specialty_text	Utterance	The chatbot asks for the user-patient to select a specialty
utter_ask_specialty_unfeat	Utterance	The chatbot asks for the user-patient to select a specialty
utter_wrong_specialty_unfeat	Utterance	The chatbot displays an error message when there are no available doctors of a specific specialty
utter_wrong_appointment_form	Utterance	The chatbot displays an error message when there are no available doctors of a specific specialty
utter_ask_pref_doctor	Utterance	The chatbot asks the user-patient, when making an appointment, to accept (or not) the proposed favorite doctor
utter_ask_doctor	Utterance	The chatbot asks for the user-patient to select a doctor
utter_ask_appointment_date	Utterance	The chatbot asks for the user-patient to select an appointment date
utter_ask_appointment_time	Utterance	The chatbot asks for the user-patient to select an appointment time
utter_ask_appointment	Utterance	The chatbot asks for the user-patient to select an appointment

Table 7: Actions for Patient stories

HTTP API

MECIS-Bot Interaction Engine communicates with MECIS-Bot Client through an HTTP API. This API is entirely defined and controller by the Rasa Framework. The only thing that the Bot developer needs to do is to specify, during the implementation phase, the TCP Port it is "listening" to.

In a conceptual modeling level, this API provides the following basic endpoints:

- **postMessage**

  Posts a message to the Interaction Engine, which may represent a user question or the user response to the last chatbot prompt message.

- **restart**

  Terminates the current story-conversation and starts a new one.

MECIS-Bot Actions Engine

Those actions that require code execution to run (i.e. the custom actions and forms contained in Table 6 and Table 7 above) are defined and executed on MECIS-Bot Actions Engine (see Section 4.2.1).

The Actions Engine has the exclusive ability to communicate with the Database Server. This means that it is possible to create actions that can exploit, in real time, all the data of the database.

In addition, it can communicate with any Web Server to use any kind of Web Services.

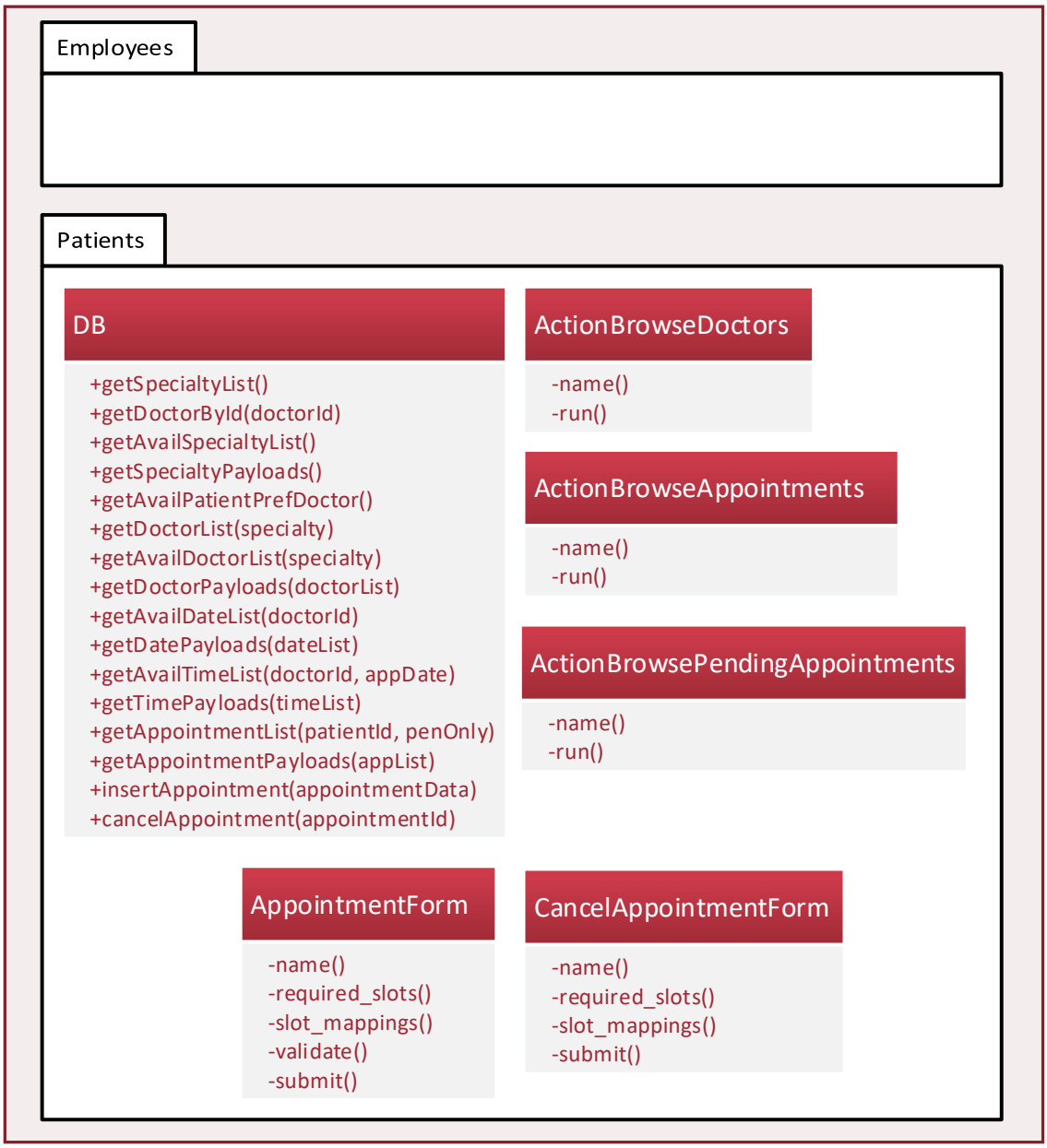
The Actions Engine is accessed only by the Interaction Engine and not by the MECIS-Bot Client. In a sense, it acts like the "actions processor" of the Interaction Engine.

Architecture

Like the Interaction Engine architecture, all the objects that make up the MECIS-Bot Actions Engine are subdivided in two independent packages (i.e. Employees and Patients).

According to Rasa Framework, the Actions Engine contains the Python code that defines all the custom actions and the forms. There must be a Python class for each custom action or form. In addition, of course, it may also contain special auxiliary classes.

So, the classes that MECIS-Bot Actions Engine contains are shown in Picture 19. It is important to note that there are no custom actions or forms for Employees. Therefore, there are no corresponding classes in their package.



Picture 19: MECIS-Bot Actions Engine Architecture (Class Diagram)

In particular, the classes that included in Patients package are described in the following table (see Table 8).

Class	Description
DB	It is a common helper class that provides database services
ActionBrowseDoctors	It defines the "action_browse_doctors". It shows – on screen - the doctors of a specialty.
ActionBrowseAppointments	It defines the "action_browse_appointments". It shows – on screen – all the patient's appointments.
ActionBrowsePendingAppointments	It defines the "action_browse_pending_appointments". It shows – on screen – all the pending patient's appointments.
AppointmentForm	It defines the "appointment_form". It executes all the necessary steps for making an appointment.
CancelAppointmentForm	It defines the "cancel_appointment_form". It executes all the necessary steps for canceling an appointment.

Table 8: MECIS-Bot Actions Engine classes for Patients

HTTP API

MECIS-Bot Actions Engine communicates with the Interaction Engine through an HTTP API. This API is entirely defined and controlled by the Rasa Framework. The only thing that the Bot developer needs to do is to specify, during the implementation phase, the TCP Port it is "listening" to.

4.4 MECIS-Bot Implementation

The implementation of MECIS-Bot is based solely on open standards, tools and languages.

4.4.1 MECIS-Bot Client

MECIS-Bot Client is completely implemented on HMTL, JavaScript and CSS. Its source code is contained inside the parent folder that contains the source code of the whole MECIS application.

MECIS is an application that has been developed as a "NetBeans 8.2" project. The name of the project is "**mecis**" and so is called the main folder containing its various source files.

MECIS-Bot UI

The MECIS-Bot UI component is essentially a JavaScript object called "**wppBot**" and is embedded in the broader MECIS client. Its source code is contained in the JavaScript file "**mecis\client\js\wpp.bot.js**" where, "**mecis**" is the main folder that contains the source code of the whole MECIS application.

In order to meet the UI requirements deriving from the system modeling principles (see Section 4.3), the MECIS-Bot UI component is based on the BotUI framework (see Section 4.2.2). Thus, creating and using an object of the BotUI class, it has total control over the client area of the chatbot.

In addition, as the MECIS-Bot UI component is responsible for communicating with the MECIS-Bot Interaction Engine, it has the obligation to access the relevant API. Therefore, to do this, it relies on AJAX technology.

Summarizing, the MECIS-Bot UI performs the following basic actions:

- **Displaying messages**
- **Getting user input**
- **Consuming the MECIS-Bot Interaction Engine API**

Displaying messages

According to the owner of the message, there are two types of messages that the chatbot should display: i) chatbot messages and ii) human messages.

Therefore, in order to show a chatbot message, MECIS-Bot uses the BotUI Message element as follows:

```
botui.message.add({
  content: 'Hello from bot.'
});
```

In order to show a human message, MECIS-Bot uses the BotUI Message element:

```
botui.message.add({
  human: true,
  content: 'Hello from human.'
});
```

Getting user input

According to the principles of conversation design (see Section 4.3.1), there are three types of input UI elements through which data can be entered by the user: i) Text Box ii) Button Group or iii) Single Selection Dropdown List.

In order to show a Text Box to the user, MECIS-Bot uses the BotUI Text Box element as follows:

```
botui.action.text({
  action: {
    placeholder: 'Your name'
  }
});
```

In order to show a Button Group to the user, MECIS-Bot uses the BotUI Button Group element as follows:

```
botui.action.button({
  action: [
    { // show a "Yes" button
      text: 'Yes',
      value: 'yes'
    },
    { // show a "No" button
      text: 'No',
      value: 'no'
    }
  ]
})
```

Finally, in order to show a Single Selection Dropdown List to the user, MECIS-Bot uses the BotUI Single Selection Dropdown List element as follows:

```
botui.action.select({
  action: {
    placeholder: "Select Language",
    value: 'TR',          // Selected value
    searchselect: false, // to act as a standard dropdown
    label: 'text',        // dropdown label variable
    options: [
                {value: "EN", text: "English"},
                {value: "ES", text: "Español"},
                {value: "TR", text: "Türkçe"},
                {value: "DE", text: "Deutsch"},
                {value: "FR", text: "Français"},
                {value: "IT", text: "Italiano"},
              ],
    button: {
      icon: 'check',
      label: 'OK'
    }
  }
})
```

Consuming the MECIS-Bot Interaction Engine API

Whenever the MECIS-Bot UI has to call a method of the MECIS-Bot Interaction Engine API it is just makes an AJAX call to the URL of the method. For a typical method, this is done as follows:

```
$.ajax({
  type: 'POST',
  url: methodURL,
  contentType: "text/plain; charset=utf-8",
  data: params,
  success: function (response) {
             // code for successful call
             ……
             ……
           },
  error: function (response) {
           // code for failed call
           ……
           ……
         }
});
```

4.4.2 MECIS-Bot Server

In general, the implementation of the MECIS-Bot Server follows the principles and directives of Rasa framework, as defined by the Rasa documentation [78].

So, according to Rasa, each Rasa project implements two servers:

- **A Rasa (main) Server**

  It is responsible for the conversation management.

- **A Rasa Actions Server**

  It defines and executes *custom actions* and *forms*.

In addition, according to the system modeling of MECIS-Bot Server (see Section 4.3.4), the MECIS-Bot Server consists of two basic components (i.e. Interaction Engine and Actions Engine). Each one of them is subdivided in two packages (i.e. Employees and Patients).

Thus, based on the above facts, two different Rasa projects must be developed for the implementation of MECIS-Bot Server, one for employees and one for patients.

Rasa projects

So, the MECIS-Bot Server is derived from two separate Rasa projects each implementing a set of two Rasa servers.

Rasa project for Employees

It is called "**mecis_e_bot**" and all of its files are contained inside the "**mecis_e_bot**" folder. Actually, it implements the following Rasa servers:

- **Rasa (main) Server for Employees**
- **Rasa Actions Server for Employees**

Rasa project for Patients

It is called "**mecis_p_bot**" and all of its files are contained inside the "**mecis_e_bot**" folder. Actually, it implements the following Rasa servers:

- **Rasa (main) Server for Patients**
- **Rasa Actions Server for Patients**

MECIS-Bot Interaction Engine

Consequently, the MECIS-Bot Interaction Engine is implemented by the following two servers.

Rasa (main) Server for Employees

It is configured to listen on TCP Port **5005**. It is implemented by the following project files:

File	Project	Content
data\nlu.md	mecis_e_bot	- The definition of the *intents* that the chatbot can identify, from the inserted human messages.
data\stories.md	mecis_e_bot	- The body of the *stories* that the chatbot can develop while interacting with human.
domain.yml	mecis_e_bot	- The *declaration* of all the *entities*, *intents* and *actions* that make up the stories. - The definition of all the *utterance actions*.

Rasa (main) Server for Patients

It is configured to listen on TCP Port **5006**. It is implemented by the following project files:

File	Project	Content
data\nlu.md	mecis_p_bot	- The definition of the *intents* that the chatbot can identify, from the inserted human messages.
data\stories.md	mecis_p_bot	- The body of the *stories* that the chatbot can develop while interacting with human.
domain.yml	mecis_p_bot	- The *declaration* of all the *entities*, *intents* and *actions* that make up the stories. - The definition of all the *utterance actions*.

MECIS-Bot Actions Engine

Similarly, the MECIS-Bot Actions Engine is implemented by the following two servers.

Rasa Actions Server for Employees

It is configured to listen on TCP Port **5055**. It is implemented by the following project files:

File	Project	Content
endpoints.yml	mecis_e_bot	- The definition of the TCP Port in which the Actions Server listens (i.e. 5055).
actions.py	mecis_e_bot	- The Python code that defines all the *custom actions* (including *forms*).

Rasa Actions Server for Patients

It is configured to listen on TCP Port **5056**. It is implemented by the following project files:

File	Project	Content
endpoints.yml	mecis_p_bot	- The definition of the TCP Port in which the Actions Server listens (i.e. 5056).
actions.py	mecis_p_bot	- The Python code that defines all the *custom actions* (including *forms*).

4.5 MECIS-Bot Deployment

As mentioned in previous sections, MECIS-Bot is a kind of add-on to the MECIS application. Therefore, for MECIS-Bot to be deployed into a machine, MECIS deployment must have been preceded (see Section 3.3.9).

However, since the MECIS-Bot Client is a component embedded in the main page of the MECIS application, the deployment of MECIS-Bot is only concerned with the parts of the MECIS-Bot Server.

4.5.1 Prerequisites

The following prerequisites must have already been installed on the deployment machine:

- ✓ MECIS Application (see Section 3.3.9)
- ✓ Anaconda3 64-bit

  Anaconda [87] is one of the most popular Python Data Science platforms.

- ✓ Visual Studio Build Tools (version 2015 or later)
- ✓ Rasa Framework 64-bit (version 1.1.8 or later)

  It should be noted that, in order for Rasa to be installed on a machine, this machine must have a Graphics Processing Unit (GPU) [88].

- ✓ MySQLdb Pyhon library

  MySQLdb is a Python library for accessing MySQL databases. Usually, it is not included in a standard Anaconda3 installation. The MySQLdb Python can be installed, using pip, as follows:

  ```
  pip install mysqlclient
  ```

4.5.2 Deployment process

The deployment material of MECIS-Bot Server is contained in the "**SourceFiles**" folder that accompanies this thesis document. It consists of the following parts:

- The "**mecis-e-bot**" folder that contains the homonymous Rasa project for Employees.
- The "**mecis-p-bot**" folder that contains the homonymous Rasa project for Patients.
- The "**MECIS-Bot Train.bat**" batch file.
- The "**MECIS-Bot.bat**" batch file.

So, the following process must be followed for the deployment of the MECIS-Bot itself:

1. Initially, the above deployment material must be copied, from the "**SourceFiles**" folder, into a folder of the deployment machine.
2. Then, using a simple text editor, the word *MECIS-Bot_MASTER_FOLDER_PATH* should be replaced, within the files "MECIS-Bot Train.bat" and "MECIS-Bot.bat", with the actual full path of the above target folder.
3. Finally, the "MECIS-Bot Train.bat" batch file must be executed in order for the Rasa projects to be prepared for operation.

Now, the only thing left to start MECIS-Bot Server is to execute the "MECIS-Bot.bat" batch file. As a result, the following Rasa Servers will start:

- ✓ Rasa (main) Server for Employees
- ✓ Rasa Actions Server for Employees
- ✓ Rasa (main) Server for Patients
- ✓ Rasa Actions Server for Patients

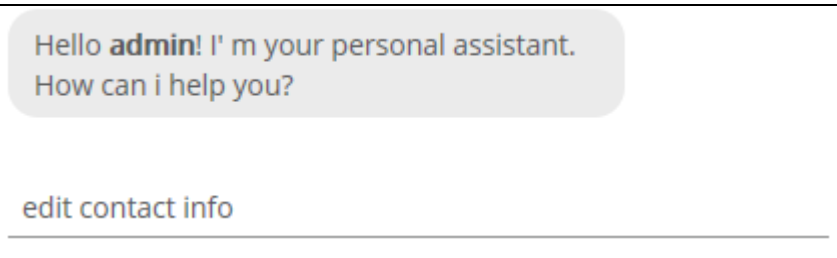
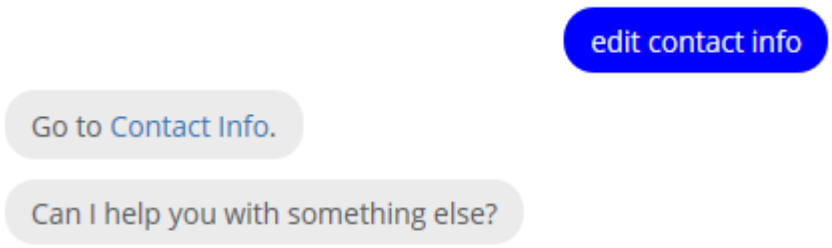
4.6 MECIS-Bot Operation and Experiments

In the software engineering industry, it is well known that one of the best ways to test the functionality of a computer system is to run a number of indicative test cases. In general, each test case attempts to verify that the basic or an alternative flow of a system use case is executed correctly. So, actually, each test case is closely related to a system use case.

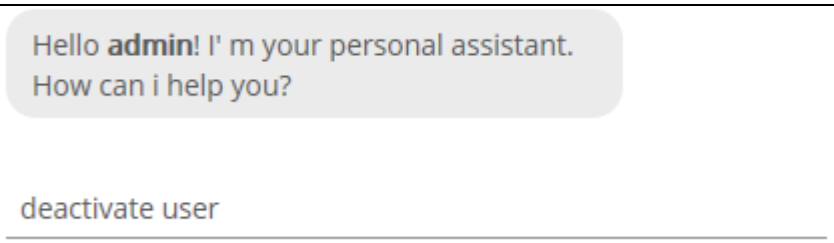
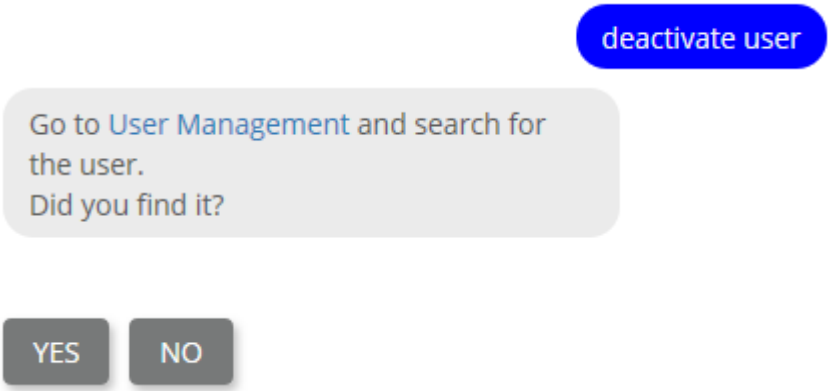
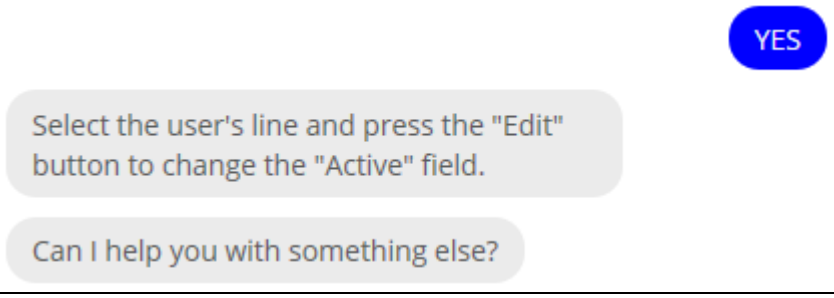
The following test cases are experimenting with the functionality of MECIS-Bot by performing various variations of the use cases of Section 4.1.

4.6.1 Test Cases for Employees

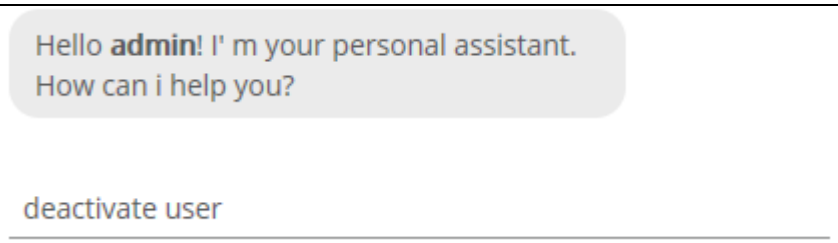
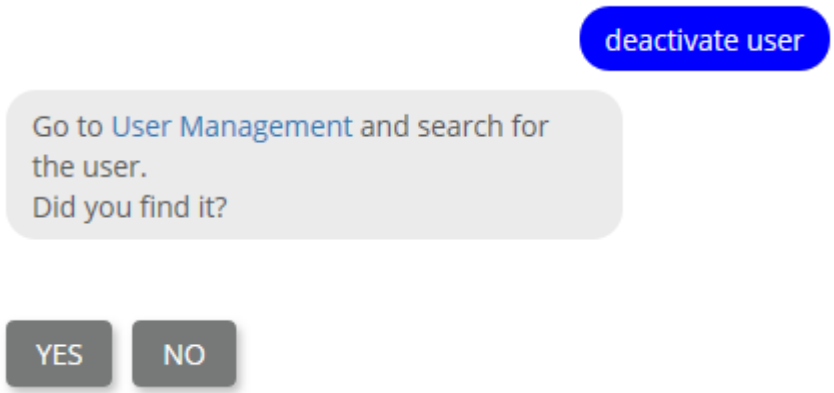
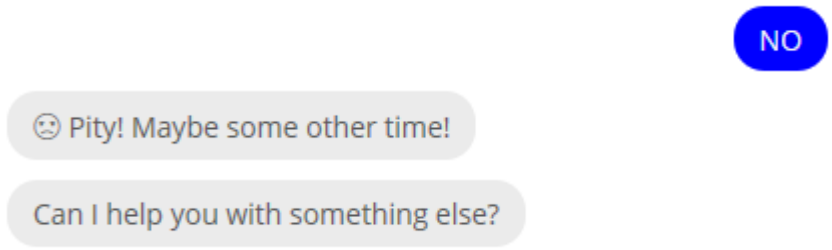
[UC11/TC1] How to edit the contact info of the medical center?

Title	Asking for help to edit the contact info of the medical center
Use case	[UC11] Making a typical "how-to" question
Execution Prerequisites	The current user has signed in as an Employee (e.g. *admin*).
Execution flow	 

[UC11/TC2] How to deactivate a user ("happy path")?

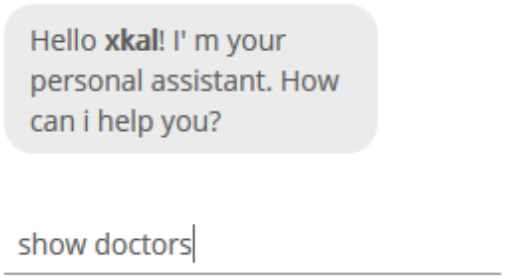
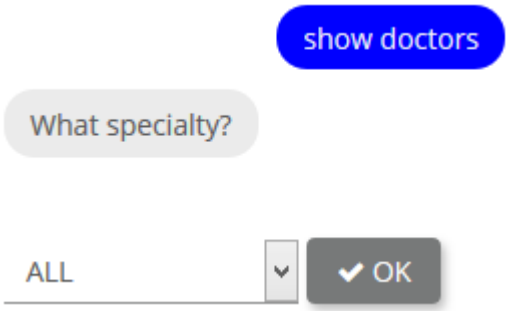
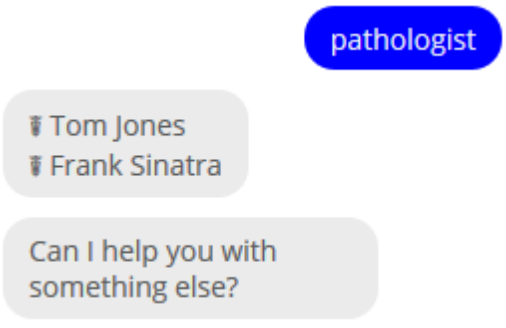
Title	Asking for help to deactivate a user ("happy path")
Use case	[UC11] Making a typical “how-to” question
Execution Prerequisites	The current user has signed in as an Employee (e.g. *admin*).
Execution flow	  

[UC11/TC3] How to deactivate a user ("unhappy path")?

Title	Asking for help to deactivate a user ("unhappy path")
Use case	[UC11] Making a typical “how-to” question
Execution Prerequisites	The current user has signed in as an Employee (e.g. *admin*).
Execution flow	  

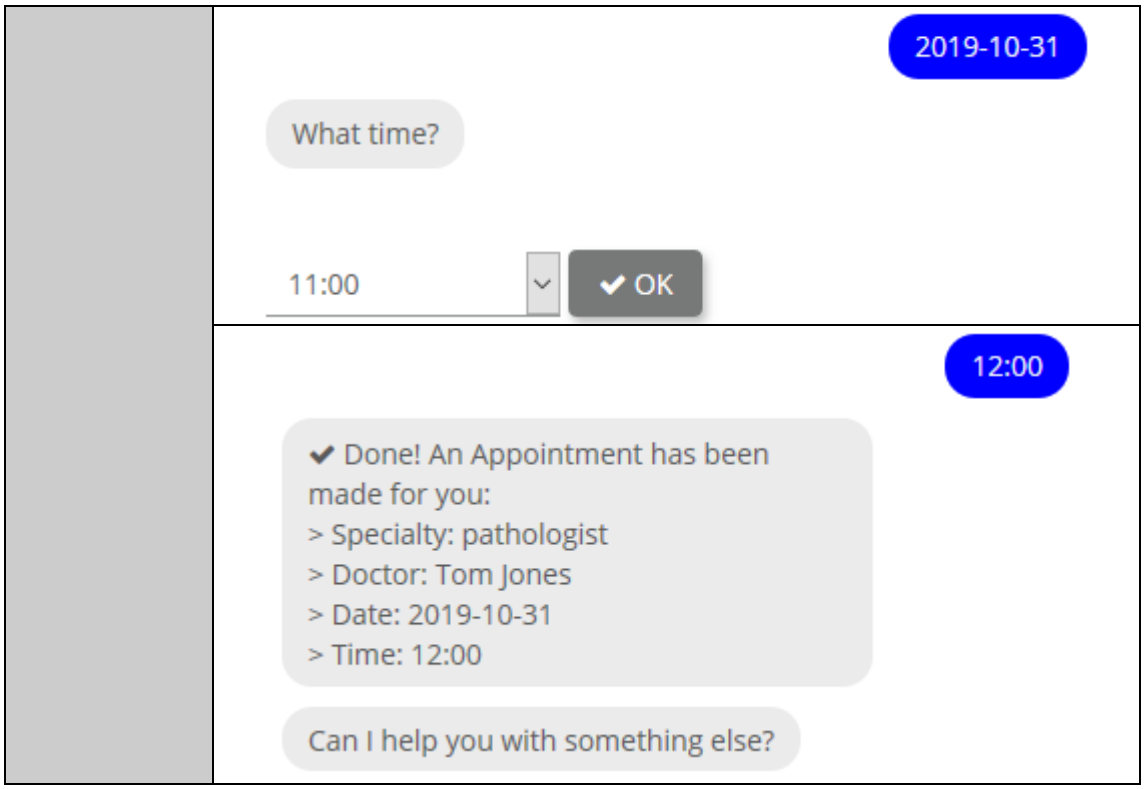
4.6.2 Test Cases for Patients

[UC21/TC1] Looking for pathologists

Title	Looking for pathologists
Use case	[UC21] Looking for doctors
Execution Prerequisites	• The database of MECIS contains one or more pathologists. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	  

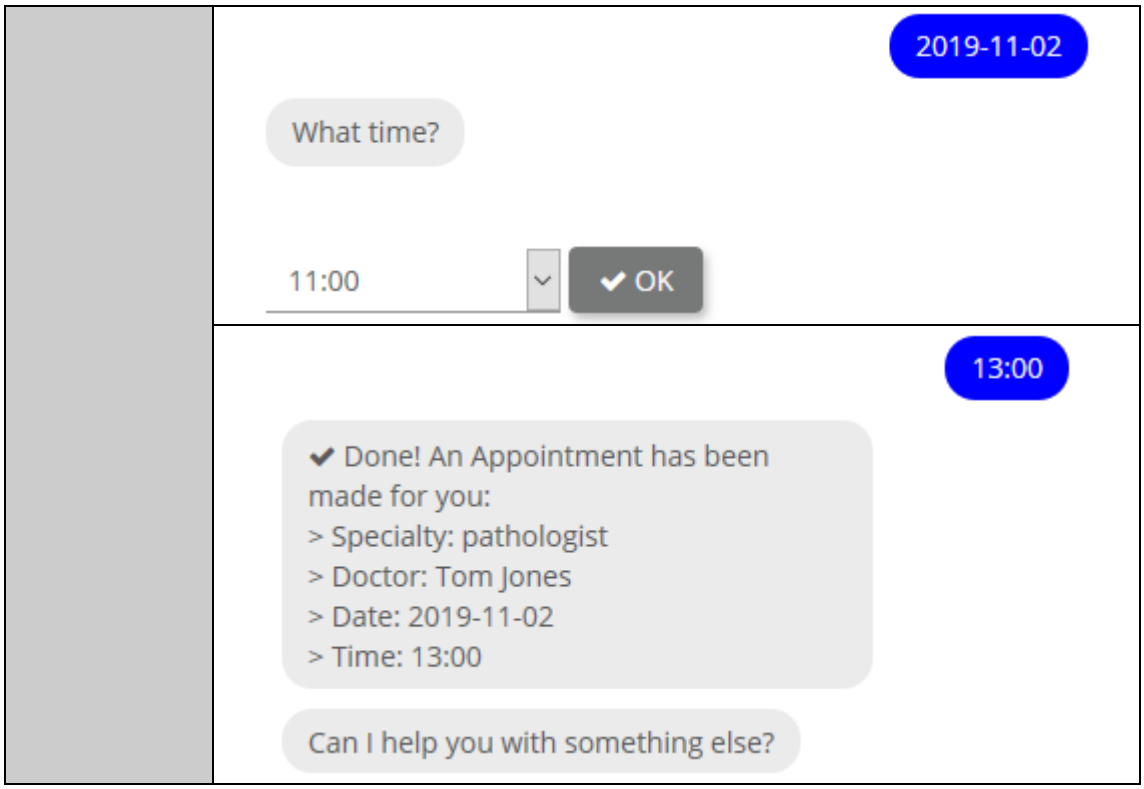
[UC22/TC1] Making an appointment with a pathologist

Title	Making an appointment with a pathologist
Use case	[UC22] Making an appointment
Execution Prerequisites	• The database of MECIS contains at least two available - for the current week – pathologists. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	

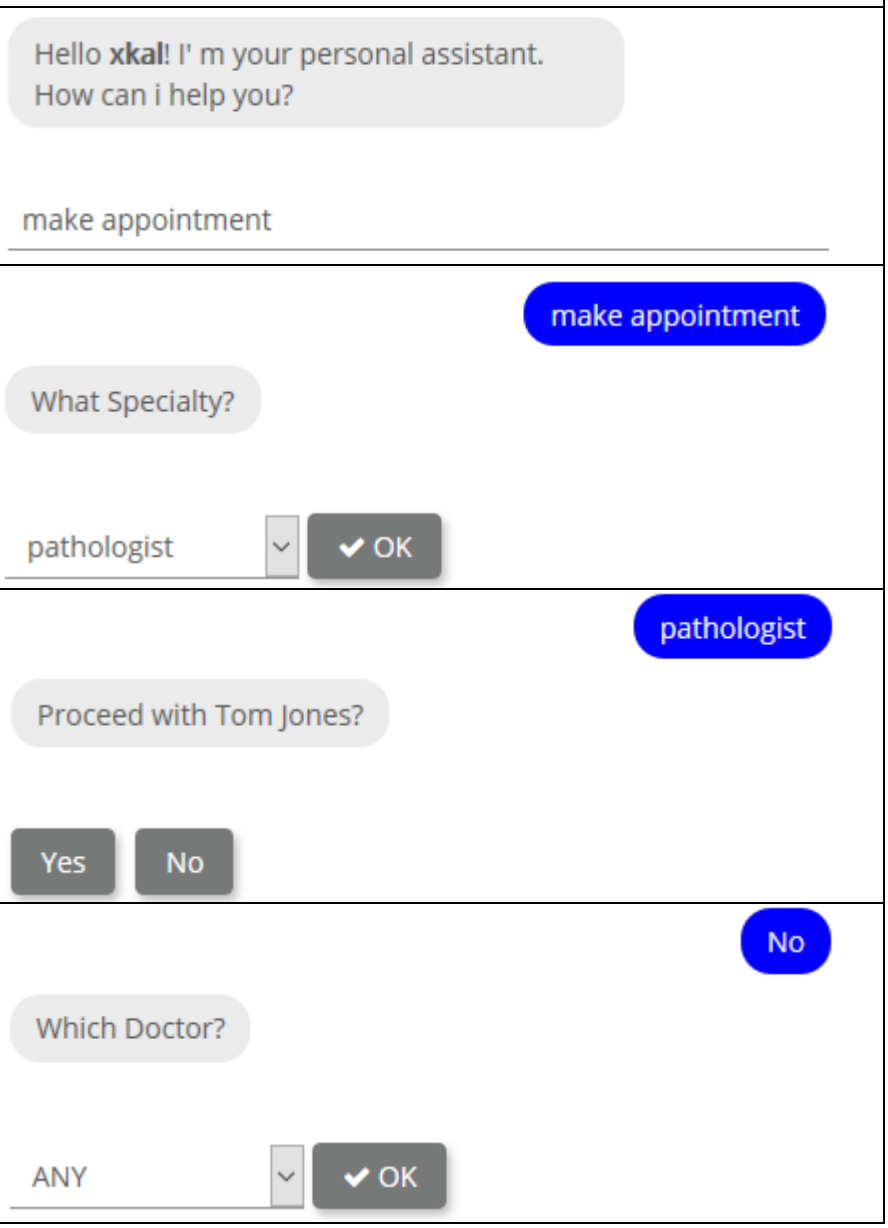

2019-10-31
What time?
11:00
OK
12:00
Done! An Appointment has been made for you:
> Specialty: pathologist
> Doctor: Tom Jones
> Date: 2019-10-31
> Time: 12:00
Can I help you with something else?

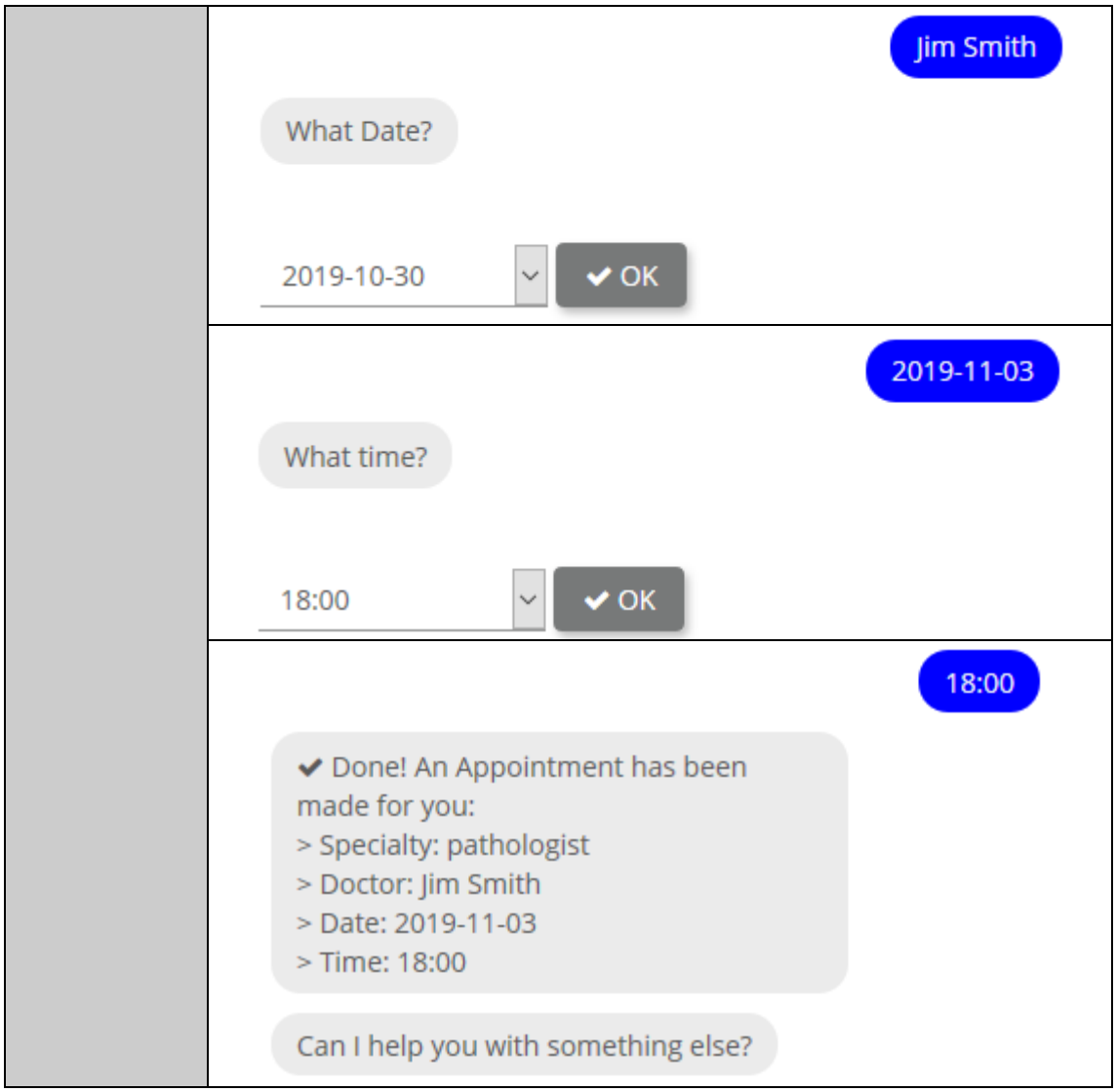
[UC22/TC2] Making an appointment with the favorite pathologist

Title	Making an appointment with the favorite pathologist
Use case	[UC22] Making an appointment
Execution Prerequisites	• The database of MECIS contains at least two available – for the current week - pathologists and the Patient has already made an appointment with one of them (pending or not). • The current user has signed in as a Patient (e.g. user).
Execution flow	

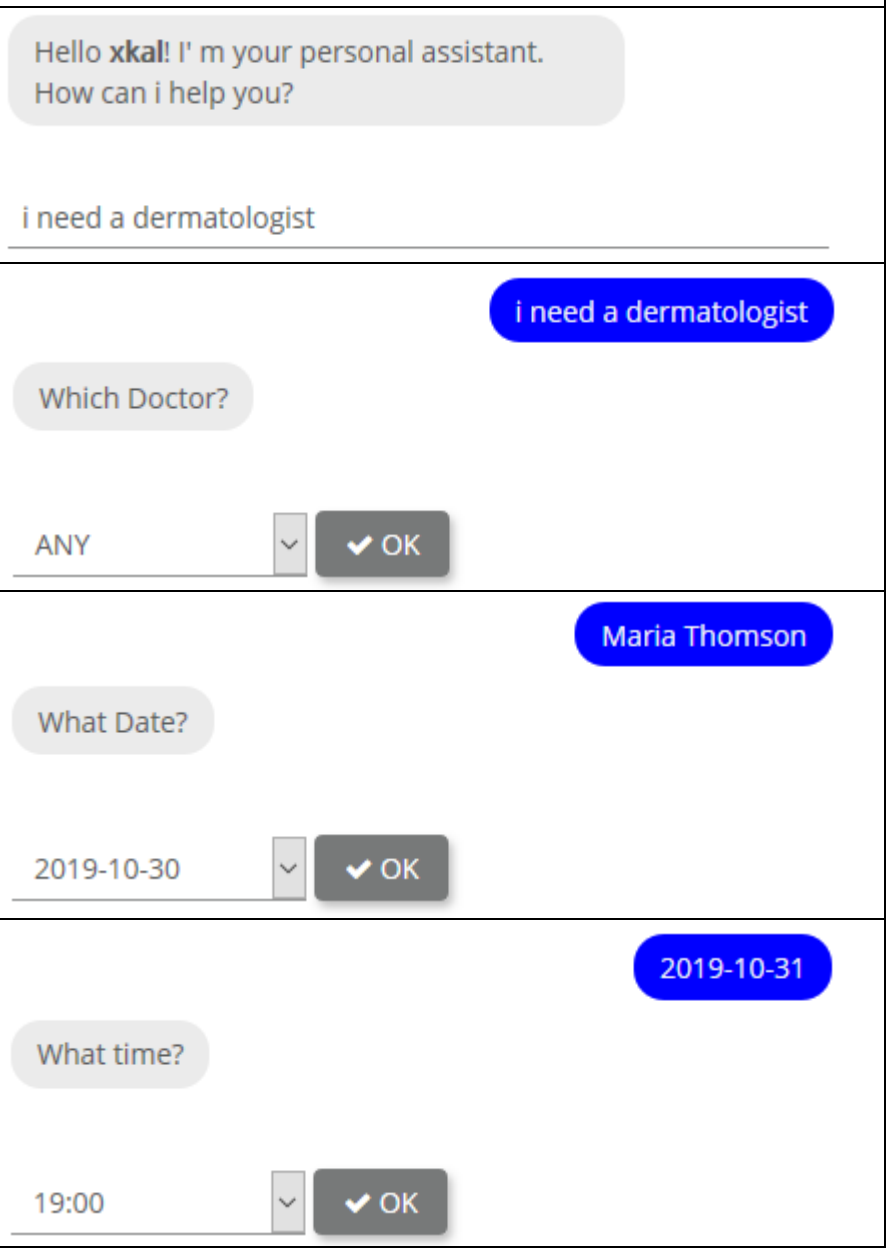

2019-11-02
What time?
11:00
OK
13:00
Done! An Appointment has been made for you:
> Specialty: pathologist
> Doctor: Tom Jones
> Date: 2019-11-02
> Time: 13:00
Can I help you with something else?

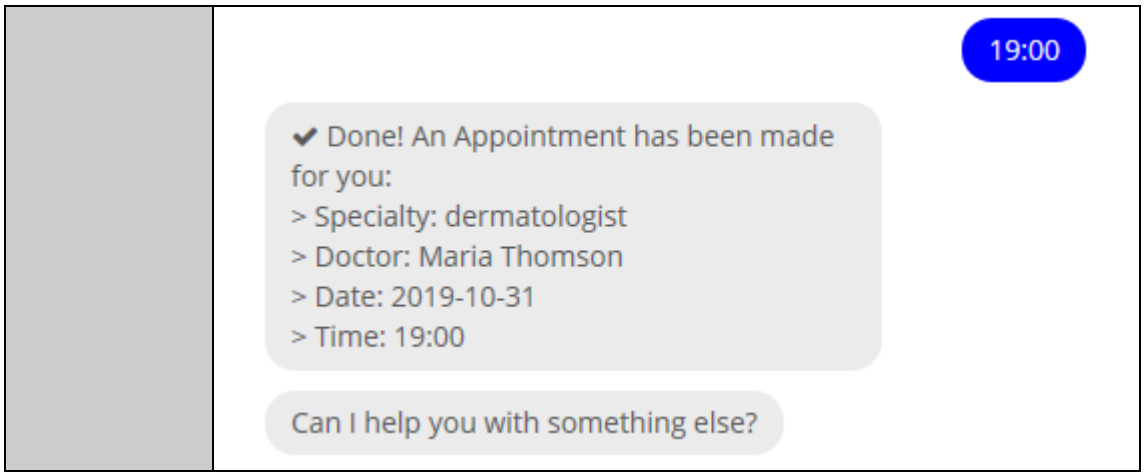
[UC22/TC3] Making an appointment with a pathologist other than the favorite one

Title	Making an appointment with a pathologist other than the favorite one
Use case	[UC22] Making an appointment
Execution Prerequisites	• The database of MECIS contains at least two available – for the current week - pathologists and the Patient has already made an appointment with one of them (pending or not). • The current user has signed in as a Patient (e.g. *user*).
Execution flow	

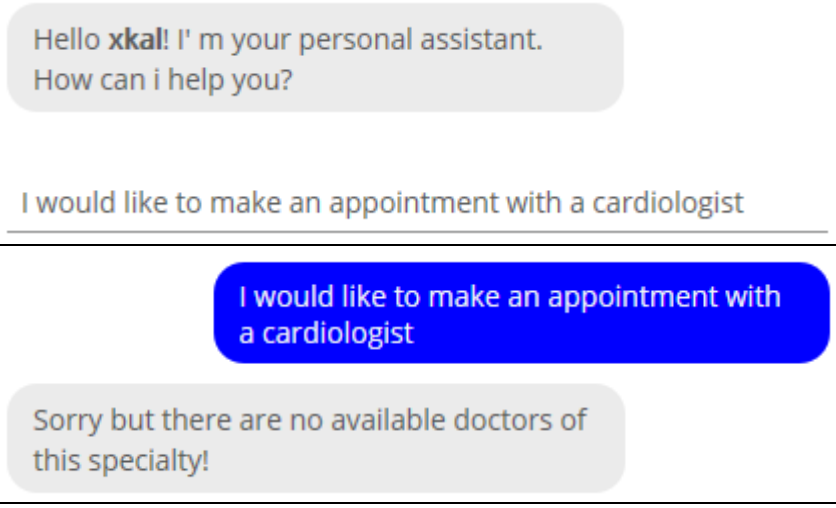

Jim Smith
What Date?
2019-10-30
OK
2019-11-03
What time?
18:00
OK
18:00
Done! An Appointment has been made for you:
> Specialty: pathologist
> Doctor: Jim Smith
> Date: 2019-11-03
> Time: 18:00
Can I help you with something else?

[UC22/TC4] Making an appointment asking for a dermatologist from the beginning of the request

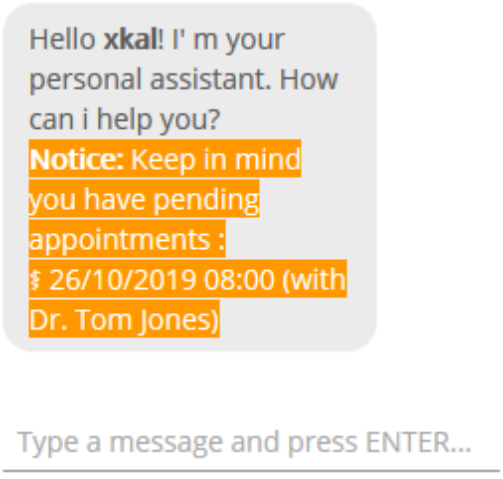
Title	Making an appointment asking for a dermatologist from the beginning of the request
Use case	[UC22] Making an appointment
Execution Prerequisites	• The database of MECIS contains at least one available - for the current week – dermatologist. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	


19:00
✔ Done! An Appointment has been made for you:
> Specialty: dermatologist
> Doctor: Maria Thomson
> Date: 2019-10-31
> Time: 19:00
Can I help you with something else?

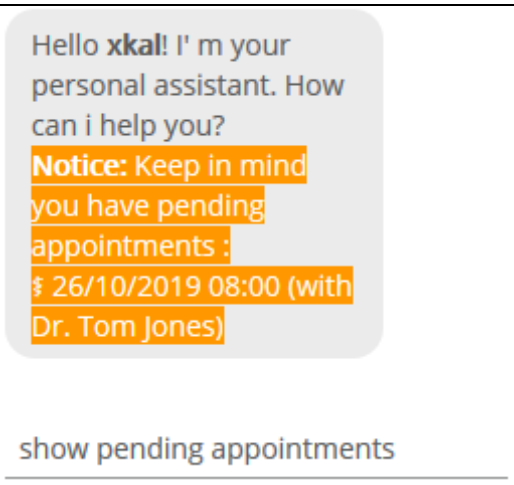
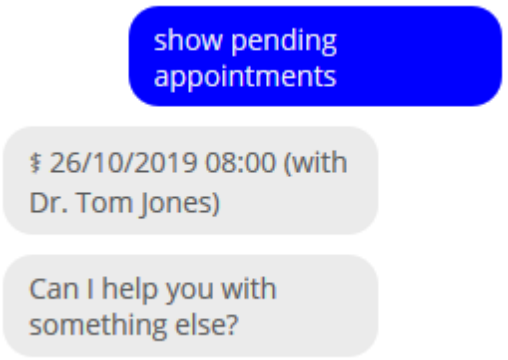
[UC22/TC5] Trying to make an appointment asking for a non-available doctor from the beginning of the request

Title	Trying to make an appointment asking for a non-available doctor (e.g. a cardiologist) from the beginning of the request ("unhappy path")
Use case	[UC22] Making an appointment
Execution Prerequisites	• The database of MECIS does not contain any available cardiologist. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	

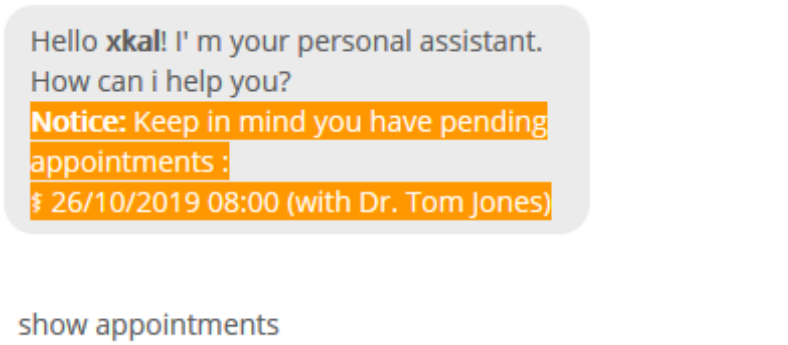
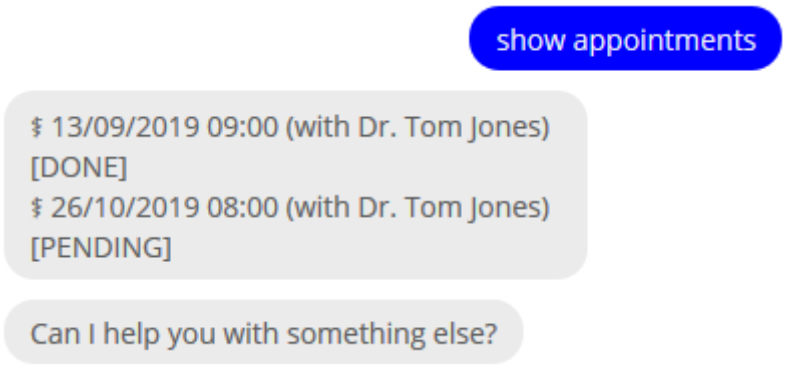
[UC23/TC1] Notifying about pending appointments

Title	Notifying about pending appointments
Use case	[UC23] Notifying about pending appointments
Execution Prerequisites	• The database of MECIS contains one or more pending appointments of the current user. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	

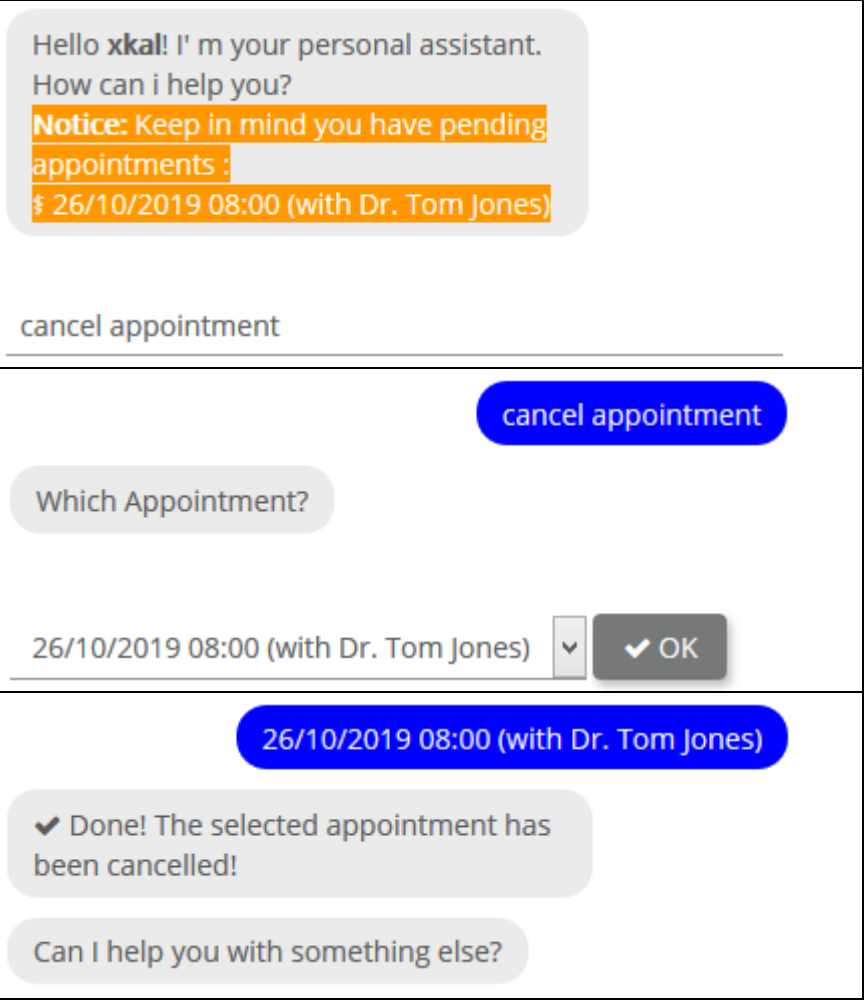
[UC24/TC1] Showing only the pending appointments

Title	Showing only the pending appointments
Use case	[UC24] Showing appointments
Execution Prerequisites	• The database of MECIS contains one or more pending appointments of the current user. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	
	

[UC24/TC2] Showing all the appointments

Title	Showing all the appointments
Use case	[UC24] Showing appointments
Execution Prerequisites	• The database of MECIS contains one or more appointments of the current user. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	 

[UC25/TC1] Canceling an appointment

Title	Canceling an appointment
Use case	[UC25] Canceling an appointment
Execution Prerequisites	• The database of MECIS contains one or more pending appointments of the current user. • The current user has signed in as a Patient (e.g. *user*).
Execution flow	

5 Conclusions

Section 1.2 presents the general intent of this thesis. Then, in Section 3.2, this intent takes the form of specific goals. Obviously, it would be very interesting to study the extent to which these goals have been achieved. In addition, throughout the course of the development of the thesis topic some interesting concerns have arisen which may, in the future, be the starting point for further studies.

5.1 General Conclusions

First of all, the basic reflection in section 3.1 has a positive answer. By studying Chapters 3 and 4, it becomes clear that:

Yes, given any large-scale application, a chatbot can be created to act as a "virtual assistant" for the users of the application and, thus, meet their needs!

In fact, MECIS-Bot manages to perfectly adapt to user needs by providing specific services for each user group (i.e. Employees and Patients). This is clear both from planning (see Section 3.4) and system analysis (see Section 4.1) where, the needs of the Employees are studied separately from the needs of the Patients.

At the same time, MECIS-Bot successfully tackles all the usual challenges that modern chatbots, working as virtual assistants, have to face (see Section 2.6.2). More specifically:

- ✓ MECIS-Bot fully satisfies the user requirements (see Section 3.4.7). Patients, for example, can meet all their needs with MECIS-Bot (see Section 4.1.2) without having to work in the standard MECIS UI or requesting human help.
- ✓ Based on its conversation design and following the interactive design style (see Section 4.3.1), MECIS-Bot strives to be user-friendly while demonstrating communication skills.

- ✓ MECIS-Bot addresses the potential "unhappy paths" that the conversation can take either by displaying appropriate messages to the user (as shown, for example, in test case UC22/TC5 of Section 4.6.2) or suggesting alternative paths (as shown, for example, in test case UC22/TC3 of Section 4.6.2).

  At the same time, the user can immediately terminate the current conversation and start a new one (as shown in use case "[UC31] Restarting the dialogue").

5.2 Goal Achievement

As stated in Section 3.2, the main goal of this thesis is to create a chatbot that would fully satisfy the first four levels of the assistant services classification (see Section 2.4.1). The cross-reference table below (see Table 9) is a clear demonstration of the achievement of this goal as it relates each category of services to the respective system use cases (see Section 4.1). In addition, the test cases in Section 4.6 demonstrate the successful implementation of these use cases.

Use Case \ Service Category	**Notification Services**	**How-to Services**	**Contextual Services**	**Personalized Services**
[UC11] Making a typical “how-to” question		X		
[UC21] Looking for doctors			X	
[UC22] Making an appointment			X	X
[UC23] Notifying about pending appointments	X			X
[UC24] Showing appointments			X	X

Table 9: Cross-reference table among services and system use cases

Similarly, the following cross-reference table (see Table 10) demonstrates how the detailed thesis goals, set out in section 3.2, were achieved. Actually, it relates each thesis goal to the respective system use cases (see Section 4.1).

Use Case \ Thesis Goal	Providing Interactive User Guidance with a Chatbot	Modeling and Performing Business Processes with a Chatbot	Providing Personalized Services with a Chatbot
[UC11] Making a typical "how-to" question	X		
[UC21] Looking for doctors		X	
[UC22] Making an appointment		X	X
[UC23] Notifying about pending appointments			X
[UC24] Showing appointments		X	X

Table 10: Cross-reference table among thesis goals and system use cases

5.3 Future Challenges

As mentioned above, MECIS-Bot manages, in accordance with its design, to provide Interactive User Guidance to the employees-users. Specifically, as shown in the execution of test cases for Employees (see Section 4.6.1), MECIS-Bot guides the user to either go to a workplace or to do something within it. Obviously, it would certainly be more convenient - for the user - if the chatbot could perform these actions on its own! Thus, user participation would be limited to data input only (whenever required).

Of course, for MECIS-Bot, the ability to provide such services far exceeds the ability to provide interactive guidance. It would also require a stronger bond between MECIS and MECIS-Bot which does not belong to the objectives of this thesis. In any case, however, it is a very interesting prospect for the future.

Another interesting perspective is the possibility of involving more than one user in business processes. Currently, according to MECIS-Bot planning, all business processes implemented by MECIS-Bot involve only one user. This, of course, is enough to meet the requirements that have been placed on the current chatbot (see Section 3.4.7).

But what if a business process involves many users? For example, suppose a process in which a patient-user makes an appointment and then an employee-user approves it. To accomplish this process, the chatbot should enable a patient-user to start a story and, for

an employee-user, to continue it from the appropriate point. This case is certainly very difficult to handle both from Rasa and from the chatbot developer! However, enriching MECIS-Bot with a feature like this would upgrade it to a complete business process engine equivalent to a classic BPM engines (see Section 2.2.1).

Bibliography

[1] F. D. Davis, User acceptance of information technology: system characteristics, user perceptions and behavioral impacts, International Journal of Man-Machine Studies, 1993.

[2] B. Shneiderman and C. Plaisant, Designing the user interface: strategies for effective human-computer interaction, Pearson Education India, 2010.

[3] T. C. O'Rourke and B. T. O'Neill, Graphical user interface, US Patent, 1994.

[4] J. J. Garrett, The Elements of User Experience: User-Centered Design for the Web and Beyond, Voices That Matters, 2010.

[5] J. Nielsen and P. Laubheimer, "Top 10 Application-Design Mistakes," 2019. [Online]. Available: https://www.nngroup.com/articles/top-10-application-design-mistakes/.

[6] L. J. Beilby, J. Zakos and G. A. McLaughlin , Chatbots, United States Patent, 2014.

[7] D. McMurrey, "User Guides," HCEXRES, [Online]. Available: https://www.prismnet.com/~hcexres/textbook/user_guides.html.

[8] M. Indulska, P. Green, J. Recker and M. Rosemann, Business Process Modeling: Perceived Benefits, Springer, 2009.

[9] T.-P. Liang, H.-J. Lai and Y.-C. Ku, Personalized Content Recommendation and User Satisfaction: Theoretical Synthesis and Empirical Findings, Journal of Management Information Systems, 2006.

[10] J. Becker, M. Rosemann and C. Von Uthmann, Guidelines of Business Process Modeling, Springer, 2000.

[11] "Business Process Model And Notation," OMG, 2011. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/.

[12] J. Hong, E. Suh, J. Kim and S. Kim, Context-aware system for proactive personalized service based on context history, ScienceDirect, 2008.

[13] A. J. Bodart and C. E. Condon, Virtual Assistant, US 2007/0043687 A1, USA Patent Application Publication, 2007.

[14] A. Nichol, "The next generation of AI assistants in enterprise," O' Reilly, 2018. [Online]. Available: https://www.oreilly.com/ideas/the-next-generation-of-ai-assistants-in-enterprise.

[15] P. Semaan, Natural Language Generation: An Overview, Journal of Computer Science & Research (JCSCR)-ISSN, 2012.

[16] E. Booker, Voice recognition enters the mainstream, Computerworld, 1994.

[17] A. Deshpande, A. Shahane, D. Gadre, M. Deshpande and P. M. Joshi, A SURVEY OF VARIOUS CHATBOT IMPLEMENTATION TECHNIQUES, International Journal of Computer Engineering and Applications, 2017.

[18] A. Turing, Computing Machinery and Intelligence, Mind, 1950.

[19] J. Weizenbaum, Computer Power and Human Reason: From Judgment to Calculation, W.H. Freeman and Company, 1976.

[20] M. Jain, P. Kumar, R. Kota and S. N. Patel , Evaluating and Informing the Design of Chatbots, Proc. of the SIGCHI CHI Conference. ACM, 2018.

[21] A. P. Chaves and M. A. Gerosa, How should my chatbot interact? A survey on human-chatbot interaction design, Cornell University, 2019.

[22] R. O. Prates, C. S. De Souza and S. D. Barbosa, Methods and tools: a method for evaluating the communicability of user interfaces, ACM, 2000. .

[23] C. S. De Souza, R. O. Prates and S. D. Barbosa, A method for evaluating software communicability, PUC-RioInf 1200, 1999.

[24] T. Grossman, G. Fitzmaurice and R. Attar, A survey of software learnability: metrics, methodologies and guidelines, ACM, 2009.

[25] F. A. Valério, T. G. Guimarães, R. O. Prates and H. Candello, Here’s What I Can Do: Chatbots’ Strategies to Convey Their, ACM, 2017.

[26] D. Gregory, H. Iris, A. David, K. Rohit and P. R. Carolyn, Towards academically productive talk supported by conversational agents, Springer, 2013.

[27] K. Morrissey and J. Kirakowski, 'Realness' in Chatbots: Establishing Quantifiable Criteria, Springer, 2013.

[28] H. Fan and M. S. Poole, What is personalization? Perspectives on the design and implementation of personalization in information systems, Journal of Organizational Computing and Electronic Commerce, 2006.

[29] K. Björkqvist, K. Österman and A. Kaukiainen, Aggression and Violent Behavior, Elsevier Science Ltd, 2000.

[30] M. Neururer, S. Schlögl, L. Brinkschulte and A. Groth, Perceptions on Authenticity in Chat Bots, Multimodal Technologies and Interaction, 2018.

[31] A. Salovaara and A. Oulasvirta, Six modes of proactive resource management: a user-centric typology for proactive behaviors, ACM, 2004.

[32] D. Tennenhouse, Proactive computing, ACM, 2000.

[33] S. . A. Abdul-Kader and J. Woods, Survey on Chatbot Design Techniques in Speech Conversation Systems, International Journal of Advanced Computer Science and Applications, 2015.

[34] C.-H. Lee, From Knowledge-Ignorant to Knowledge-Rich Modeling : A New Speech Research Paradigm for Next-Generation ASR, Center of Signal and Image Processing Georgia Institute of Technology, 2004.

[35] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, Natural Language Processing (Almost) from Scratch, Journal of Machine Learning Research, 2011.

[36] D. Khurana, A. Koli, K. Khatter and S. Singh, Natural Language Processing: State of The Art, Current Trends and Challenges, Accendere Knowledge Management Services Pvt. Ltd., India.

[37] D. J. Stoner, L. Ford and M. Ricci, Simulating Military Radio Communications Using Speech Recognition and Chat-Bot Technology, The Titan Corporation Orlando, Florida, 2003.

[38] N. A. Ahmad, M. H. C. Hamid, A. Zainal, M. F. A. Rauf and Z. Adnan, Review of Chatbots Design Techniques, International Journal of Computer Applications, 2018.

[39] R. Budiu, "The User Experience of Chatbots," Nielsen Norman Group, 2018. [Online]. Available: https://www.nngroup.com/articles/chatbots/.

[40] LogMeIn, Inc., "Conversational AI - Bold 360," LogMeIn, Inc., [Online]. Available: https://www.bold360.com/features/conversational-ai.

[41] Bank of Montreal, "Chatbots - Way to Bank," BMO, [Online]. Available: https://www.bmo.com/main/personal/chatbot/#what-is-chatbot.

[42] S. Ghose and J. J. Barua, Toward the implementation of a topic specific dialogue based natural language chatbot as an undergraduate advisor, International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, 2013.

[43] H. F. Schantz, The history of OCR, optical character recognition, Recognition Technologies Users Association, 1982.

[44] M. Heilman and N. A. Smith, Question Generation via Overgenerating Transformations and Ranking, Language Technologies, Institute School of Computer Science, Carnegie Mellon University, 2009.

[45] "AIML," [Online]. Available: http://www.aiml.foundation/.

[46] L. Pichponreay, J.-H. Kim, C.-H. Choi, K.-H. Lee and W.-S. Cho, Smart answering Chatbot based on OCR and Overgenerating Transformations and Ranking, IEEE, 2016.

[47] C. Lei, H. Shaohan, W. Furu, T. Chuanqi, D. Chaoqun and Z. Ming, SuperAgent: A Customer Service Chatbot for E-commerce Websites, Proc. of the 55th Annual Meeting of Assoc. for Computational Linguistics-System Demonstrations, 2017.

[48] "Erica," [Online]. Available: https://promo.bankofamerica.com/erica/.

[49] "Charlie," [Online]. Available: https://www.chatbots.org/virtual_agent/charlie/.

[50] "Microsoft Bot Framework," [Online]. Available: https://dev.botframework.com/.

[51] "Facebook for Developers," [Online]. Available: https://developers.facebook.com/products/messenger.

[52] "Google Assistant," [Online]. Available: https://assistant.google.com/.

[53] "Amazon Lex," [Online]. Available: https://aws.amazon.com/lex/.

[54] D. Bobbarjung, M. Mathihalli, M. Tiwari, K. MacDonald and R. Narasimhan Raj, CHATBOT SKILLS SYSTEMS AND METHODS, United States Patent Application Publication, 2019.

[55] M. Portela and C. Granell-Canut, A new friend in our smartphone?: observing interactions with chatbots in the search of emotional engagement, 2017: Proc. of the Int. Conf. on Hum. Comput. Interact. ACM.

[56] T. L. Vu, K. Z. Tun, C. Eng-Siong and R. E. E , Online FAQ Chatbot for Customer Support.

[57] S. Quarteroni and S. Manandhar, A chatbot-based interactive question answering, Decalog, 2007.

[58] K. Kuligowska, Commercial chatbot: performance evaluation, usability metrics and quality standards of embodied conversational agents, 2015.

[59] E. Ayedoun, Y. Hayashi and K. Seta, Can Conversational Agents Foster Learners Willingness to Communicate in a Second Language?: effects of communication strategies and affective backchannels, Proceedings of the 25th International Conference on Computers in Education, 2017.

[60] P. B. Brandtzaeg and A. Følstad, Why people use chatbots, Springer, 2017.

[61] D. Duijst, Can we Improve the User Experience of Chatbots with Personalisation, Master's thesis. University of Amsterdam, 2017.

[62] W. Duijvelshoff, Use-Cases and Ethics of Chatbots on Plek: a Social Intranet for Organizations, Workshop On Chatbots And Artificial Intelligence, 2017.

[63] H.-y. Shum, X.-d. He and D. Li, From Eliza to XiaoIce: challenges and opportunities with social chatbots, Front. Inf. Technol. Electron. Eng, 2018.

[64] K. K. Fitzpatrick, A. Darcy and D. Vierhile, Delivering cognitive behavior therapy to young adults with symptoms of depression and anxiety using a fully automated conversational agent (Woebot): a randomized controlled trial, JMIR mental health, 2017.

[65] Y. Hayashi, Social Facilitation Effects by Pedagogical Conversational Agent: Lexical Network Analysis in an Online Explanation Task, Proc. of the IEDMS, 2015.

[66] P. B. Brandtzaeg and A. Følstad, Chatbots: changing user needs and motivations, Interactions, 2018.

[67] E. Luger and A. Sellen, Like Having a Really Bad PA: The Gulf between User Expectation and Experience of Conversational Agents, Proc. of the SIGCHI CHI Conference. ACM, 2016.

[68] R. Jiang and R. E. Banchs, Towards Improving the Performance of Chat Oriented Dialogue System, 2017: Proc. of the IALP. IEEE.

[69] I. Maslowski, D. Lagarde and C. Clavel, In-the-wild chatbot corpus: from opinion analysis to interaction problem detection, International Conference on Natural Language and Speech Processing, 2017.

[70] V. Q. Liao, M. M. Hussain, P. Chandar, M. Davis, M. Crasso, D. Wang, M. Muller, S. N. Shami and G. Werner, All Work and no Play? Conversations with a Question-and-Answer Chatbot in the Wild, Proc. of the SIGCHI CHI Conference, 2018.

[71] B. Reeves and C. Nass, The Media Equation: How people treat computers, television, and new media like real people and places, 1996: CSLI Publications and Cambridge university press.

[72] B. Fogg, Computers as persuasive social actors, Persuasive Technology, 2003.

[73] J. Forlizzi, J. Zimmerman, V. Mancuso and S. Kwak, How interface agents affect interaction between humans and computers, Proc. of the Conf. on DPPI. ACM, 2007.

[74] S. Y. Lee and J. Choi, Enhancing user experience with conversational agent for movie recommendation: Effects of self-disclosure and reciprocity, Int J Hum Comput Stud.103, 2017.

[75] I. M. Thies, N. Menon, S. Magapu, M. Subramony and J. O'neill, How do you want your chatbot? An exploratory Wizard-of-Oz study with young, urban Indians, IFIP Conf. on Hum. Comput. Interact. Springer, 2017.

[76] S. Burbeck, Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC), ParcPlace Systems, Inc, 1992.

[77] "Rasa Stories," RasaStories, [Online]. Available: https://rasa.com/docs/rasa/core/stories/#.

[78] "Rasa," [Online]. Available: https://rasa.com/.

[79] D. Braun, A. Hernandez-Mendez, F. Matthes and M. Langen, Evaluating Natural Language Understanding Services for Conversational Question Answering Systems, SIGDIAL Conference, 2017.

[80] "Rasa NLU," [Online]. Available: https://rasa.com/docs/rasa/nlu/about/.

[81] "Rasa Core," [Online]. Available: https://rasa.com/docs/rasa/core/about/#.

[82] A. Nichol, "A New Approach to Conversational Software," Rasa, [Online]. Available: https://medium.com/rasa-blog/a-new-approach-to-conversational-software-2e64a5d05f2a.

[83] "Rasa Architecture," [Online]. Available: https://rasa.com/docs/rasa/user-guide/architecture/.

[84] "BotUI," [Online]. Available: https://docs.botui.org/.

[85] "Vue," [Online]. Available: https://vuejs.org/.

[86] "Rasa Actions," [Online]. Available: https://rasa.com/docs/rasa/core/actions/#.

[87] "Anaconda," 2016. [Online]. Available: https://www.anaconda.com/.

[88] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips, GPU Computing, Proceedings of the IEEE, 2008.

Glossary